

Glob.ocx - Programmer's Reference

Confidential

Page 1

08/07/00

Overview

Glob.ocx is a custom control developed to permit applications to communicate through shared memory. The term **GLOB** was coined to refer to *Global Objects*. Globs contain data, type information, some user-defined areas, and a symbolic name, or label. This custom control provides a standard interface for creating, removing, and accessing shared memory and Globs within shared memory. **The current implementation assumes that the applications are well-behaved as far as working together goes. This assumes that one application will not close or redefine the memory-mapped file (MMF) while others are accessing it.**

Location of control: _____

Control loaded at Base Address: 3000 0000 (hex)_____

Memory-Mapped Files

Memory-mapped files (MMF) are the only method available in Windows NT for applications to share memory. NT allows the user to map a file to a memory space, and to then perform either I/O operations or pointer-based operations on the memory. (MMF and Globs work equally well under Windows 95)

GLOB-structured Memory-Mapped Files

When a Glob control is initialized (when an instance of the Glob class is initialized) it opens the MMF and applies an overall structure to it. The first 172 bytes of the MMF contain the following information

Byte Offset	Type	N Bytes	Description
0	Integer	4	Size of the memory-mapped file
4	Integer	4	Offset of the first Glob position in the MMF
8	Integer	4	Offset of the next available Glob position in the MMF. (This may not always indicate the next position to be used however – if Globs are erased, others may be created by reusing the erased space)
12	integer	4	Read only flag
16	integer	4	Runtime reference count.
20	integer	4	Number of 4 byte Notify Maps in each Glob.
24	byte	40	unused space.
64	Integer	Variable	list of window handles for the notification process.

Glob.ocx - Programmer's Reference

Confidential

Page 2

08/07/00

The Glob control creates a structure in the MMF which contains a name, some parameters, and data. The following is the current structure of a Glob.

Byte Offset	Name	Type -	Byte Size	
0	Size	Long	4	Size of Glob, data, and notify maps
4	GlobName	BYTE	16	ASCII name of Glob
20	Dim2Size	Short	2	2nd dimension
22	Dim1Size	Short	2	1st dimension
24	Elementsize	Short	2	byte size of each array element
26	Type	Short	2	type of array element
28	Extra	Short	2	User-defined parameter
30	Command	Short	2	Command to the device
32	Status	Short	2	Status from the device
34	DataSize	Short	2	Actual data area used
36	UnitsIndex	Long	4	Glob index of a Unit-of-Measure label
40	Link	Long	4	Relative link to another Glob. Add to current position to get new position. Links are user-defined.
44	NotifyMap	Long	4	offset in the Glob where the notify maps can be found. The notify maps start right after the data.
48	Data	Any	var.	Data area
48+data size	NotifyMaps	Long	var.	Notification handle maps for the notification process.

GLOB.OCX

The current implementation of Glob.ocx makes the following assumptions:

The Memory-mapped file created is named "C:\GLOBMMF", this is the default name if a filename is not specified in the FileName property.

This MMF starts as 8,196 bytes in length.

BASIC GLOB METHODS AND PROPERTIES

Glob/Blob compatibility.

The Glob control is based on the Blob control and has the following properties in common with Blobs. Globbs can replace Blobs in any application without modifying the source code in any way other than to globally replace "Blob" with "Glob".

Note: The MMF files created by the Blob control are NOT compatible with the Glob control.

Code Examples.

All code examples are in Visual Basic unless otherwise specified. For more information on inserting custom controls into a Visual C++ (MFC) application, see the help entry on *ActiveX control containers: Programming ActiveX controls*.

AddNew method

Description

Adds a new Glob to the memory-mapped file.

Syntax

Glob.AddNew GlobName, UnitsIndex, Dim2Size, Dim1Size, ElementSize, Type, Extra

Remarks

GlobName: the 1-16 character name of the Glob. If a Glob by this name already exists, that one is erased and its space becomes available for reuse.

UnitsIndex: Offset in the MMF of a unit-of-measure label. 0 if none.

Dim2Size: Size of the second dimension in the data array. All *data* in a Glob is referenced through a zero-based 2-dimensional array, using the *Value* property.

Dim1Size: Size of the first dimension in the data array.

ElementSize: Size of each data element. (Long integers are 4-bytes, Short Integers are 2 bytes, and Bytes are 1 byte.) Element size is used along with *Dim1Size* and *Dim2Size* to determine the size of the data area.

Type: User-defined value.

(USP Application note: values < 5 are the number of decimal places to assume for integer values - a *Type* value of 3 means that the value 123456 represents 123.456. Another *Type* value in use is -1, meaning that the Glob is a non-data label Glob.)

Extra: User-defined value. (USP Application note: this will be used to track the actual length of a waveform during print interactions)

Note: *The current implementation guards against memory collisions. However it is not recommended that add or erase operations be used in a multiple-application environment unless you are certain that only one thread will be performing these operations on the MMF.*

Note: *(4/24/98) If the AddNew method causes the MMF to expand and there are any other globs in existence in the system (not just the current application) the MMF may not be remapped properly and a memory access violation will occur. To avoid this problem, only add globs when it is certain that there are no other globs accessing the same MMF.*

See Also *AddNewEx*

Example

```
Glob1.AddNew "RPM",0,1,1,4,0,0
```

AddNewEx method

Description

Add a new Glob and a companion unit-of-measure Glob. If the specified unit-of-measure Glob already exists, no new unit-of-measure Glob will be added.

Syntax

Glob.AddNewEx *GlobName*, *UnitsName*, *Dim2Size*, *Dim1Size*, *ElementSize*, *Type*, *Extra*

Remarks

GlobName: the 1-16 character name of the Glob. If a Glob by this name already exists, that one is erased and its space becomes available for reuse.

UnitsName: 1..16 character name of the unit-of-measure Glob. If a unit-of-measure Glob with this name already exists, no new unit-of-measure Glob will be added.

Dim2Size: Size of the second dimension in the data array. All data in a Glob is referenced through a zero-based 2-dimensional array, using the *Value* property.

Dim1Size: Size of the first dimension in the data array.

ElementSize: Size of each data element. (Long integers are 4-bytes, Short Integers are 2 bytes, and Bytes are 1 byte.) Element size is used along with *Dim1Size* and *Dim2Size* to determine the size of the data area.

Type: User-defined value. (USP Application note: values < 5 are the number of decimal places to assume for integer values. A *Type* value of 3 means that the value 123456 represents 123.456.

Extra: User-defined value. (USP Application note: this will be used to track the actual length of a waveform during print interactions)

Note: *The current implementation guards against memory collisions. However it is not recommended that add or erase operations be used in a multiple-application environment unless you are certain that only one thread will be performing these operations on the MMF.*
(see *AddNew* method)

See Also *AddNew*

Example

```
Glob1.AddNewEx "HC", "PPM", 0, 0, 4, 0, 0
```

AvailSize property

Description

Returns the number of bytes remaining in the Memory-mapped file. (Read Only)

Syntax

Longval = *Glob*.AvailSize

Remarks

The available size in the memory-mapped file will change as new Globbs are added. If a Glob forces the file to resize itself larger, the change will be reflected in the FileSize and the AvailSize properties.

See Also

FileSize

Example

```
Label1.Caption = Glob1.AvailSize
```

GlobIndex property

Description

Returns or sets the offset from the beginning of the Memory-mapped file. If Globindex is changed the location of the Glob will change, so care must be used.

Syntax

Value = *Glob*.GlobIndex

Remarks

In the current implementation, values cannot be less than 40

Example

```
Glob1.GlobName = "CO2"           ' attempt to 'link' with MMF
If Glob1.GlobIndex = -1 Then      ' not found, create it
    Glob1.AddNew "CO2", 0, 1, 1, 4, 0, 0
End If
```

GlobName property

Description

The name of the Glob. Setting a value in the GlobName property causes the control to seek a Glob by that name, and 'link' up its properties etc. if it is found.

Syntax

Glob1.GlobName = <string>

Remarks

Globs currently have names of 16 characters or fewer. If a Glob is found, the properties will thereafter reflect the values in the Memory-mapped file. If a Glob is not found by the name given, references to the GlobName property will return an empty string, and access to other properties will return the following values:

GlobIndex	-1
GlobPtr	0
Dim2Size	-1
Dim1Size	-1
Command	-1
Status	-1

See Also

Example

```
Glob1.GlobName = "CO2"           ' attempt to 'link' with MMF
If Glob1.GlobIndex = -1 Then      ' not found, create it
    Glob1.AddNew "CO2", 0, 1, 1, 4, 0, 0
End If
```

GlobPtr property

Description

32-bit pointer value pointing to the head of the current Glob in the memory-mapped file.

Syntax

Longval = *Glob*.GlobPtr

Remarks

In Windows NT, this pointer is valid only within the current process. It is intended for use with C or C++ programs which can take advantage of pointers.

See Also

DataPtr

Example

```
Dim L as Long
L = Glob1.GlobPtr ' get in-process address of Glob
```

CloseMMF method

Description

Unmaps the current memory view and closes the Memory-mapped file. Returns the status from closing the file.

Syntax

status = Glob.CloseMMF

Remarks

Note: Do not use in a multiple-application environment where other threads or processes may be using the current MMF.

Command property

Description

User-defined 16-bit value.

Syntax

Glob.Command = 16-bit-value

Remarks

Intended for use in conjunction with **Status** property for device control. By convention, the application 'owns' the **Command** property (can modify it), while the device handler must only read it.

See Also

Status

Example

```
Glob1.Command = READ_ONCE           '
While Glob1.Status <> STATUS_COMPLETE ' done yet?
    DoEvents                         ' wait
Wend
Label1.Caption = Glob1.Value(0, 0)    ' display value
```

DataPtr property

Description

32-bit pointer value pointing to the data area of the current Glob in the memory-mapped file.

Syntax

Longval = Glob.DataPtr

Remarks

This pointer is valid only within the current process. It is intended for use with C or C++ programs which can take advantage of pointers.

See Also

GlobPtr

Example

```
Dim L as Long
L = Glob1.DataPtr ' get in-process address of Glob's data
```

DataSize property

Description

By convention: the actual amount of the data area in use. This is used to indicate how many bytes of the data area contain valid data – from 0 to (Dim2Size * Dim1Size * ElementSize)-1.

Syntax

Value = Glob.DataSize

See Also

Dim2Size, Dim1Size, ElementSize

Example

```
J% = Glob1.DataSize
```

Dim1Size property

Description

Represents the second dimension in the 2-dimensional array structure of Glob data.

Syntax

Value = *Glob*.Dim1Size

Remarks

Changing this value for a Glob that already contains data will affect the calculations of the location of each data element.

See Also

Dim2Size

Example

```
Dim d2 as long, d1 as long
\
\ clear my per cylinder waveform array
\
Glob1.GlobName = "WAVES/CYL"           \ link up
For d2 = 0 to Glob1.Dim2Size - 1       \ loop through
    For d1 = 0 to Glob1.Dim1Size - 1    \ ditto
        Glob1.Value( d2, d1 ) = 0     \ zero an element
    Next d1
Next d2
```

Dim2Size property

Description

Represents the first dimension in the 2-dimensional array structure of Glob data.

Syntax

Value = *Glob*.Dim2Size

Remarks

Changing this value for a Glob that already contains data will affect the calculations of the location of each data element.

See Also

Dim1Size

Example

```
Dim d2 as long, d1 as long
\
\ clear my per cylinder waveform array
\
Glob1.GlobName = "WAVES/CYL"           \ link up
For d2 = 0 to Glob1.Dim2Size - 1       \ loop through
    For d1 = 0 to Glob1.Dim1Size - 1    \ ditto
        Glob1.Value( d2, d1 ) = 0     \ zero an element
    Next d1
Next d2
```


ElementSize property

Description

Retrieves or sets the number of bytes in a single data element. A Glob's data area is made up of a two-dimensional array of elements of size *ElementSize*.

Syntax

Value = *Glob*.ElementSize

Remarks

Changing this value for a Glob that already contains data will affect the calculations of the location of each data element and its accessed width.

See Also

Dim2Size, Dim1Size

Example

```
' Read a 32-bit integer as 4 8-bit bytes
Glob1.GlobName = "MY_LONG_INTEGER"
Glob1.Dim1Size = 4                ' 4 elements
Glob1.ElementSize = 1            ' 1 byte each
Q1 = Glob1.Value( 0, 0 )         ' read lsb
Q2 = Glob1.Value( 0, 1 )
Q3 = Glob1.Value( 0, 2 )
Q4 = Glob1.Value( 0, 3 )         ' read msb
```

Erase method

Description

Clears the contents of the current Glob, including all formatting information and data. The Glob can no longer be referenced.

Syntax

Glob.Erase

Remarks

The erased area will be reused for new Globs being added if the new Globs will fit. Erasing a Glob and then adding it again immediately will most likely reuse the same area if the overall size of the new Glob is the same as or smaller than the original.

Note: The current implementation guards against memory collisions. However it is not recommended that add or erase operations be used in a multiple-application environment unless you are certain that only one thread will be performing these operations on the MMF.

See Also

AddNew, AddNewEx, EraseMMF

Example

```
Glob1.GlobName = "RPM"           ' link to the "RPM" Glob
Glob1.Erase                      ' clear it out (delete it)
```

EraseMMF method

Description

Clears the entire Memory-mapped file and resets its internal pointers to initial values. This produces a "new" MMF.

Syntax

`Glob.EraseMMF`

Remarks

Any Globs currently linked to the MMF will contain invalid information after this operation.

Note: Do not use in a multiple-application environment unless you understand the impact on other applications.

See Also

Erase

Example

```
Glob1.EraseMMF          ` pffft, you're history
```

Extra property

Description

User-defined 16-bit integer.

Syntax

`value = Glob.Extra`

`Glob.Extra = value`

Example

```
` Tell print module to only use 100 bytes from waveform
Glob1.GlobName = "SECONDARY PARADE"
Glob1.Extra = 100
```

FileSize property

Description

Returns the current size of the MMF. Read-only.

Syntax

`Value = Glob.FileSize`

Remarks

The FileSize may change if a Glob is added to the MMF that will not fit in the current file.

See Also

FileName

FileName property

Glob.ocx - Programmer's Reference

Confidential

Page 11

08/07/00

Description

Returns or sets the file name of the current Memory-mapped file.

Syntax

String = *Glob.FileName*
Glob.FileName = <string>

Remarks

In the current implementation, the *FileName* is predefined as "C:\GLOBMMF". Changing this name will create a new MMF. The *FileName* may be a relative or absolute path. The *FullPath* property will contain the fully qualified pathname to the MMF. In general, if a full path designation is not specified in the *FileName* property then the *FullPath* will be generated by adding the current directory to the filename.

Note: *it is recommended that "C:\GLOBMMF" not be used for any real application data. Use a unique name for each new application that uses MMF.*

See Also

FileSize

Example

```
`assume our application directory is C:\myapp  
  
Glob1.FileName = "MYMMF"  
`the FullPath property will contain "C:\myapp\MYMMF"  
  
Glob1.FileName = "C:\MYMMF"  
`the FullPath property will contain "C:\MYMMF"
```

GetFirstGlob method

Description

Reads the first Glob in the Memory-mapped-file. Returns a 32-bit pointer in the MMF if found, otherwise returns 0.

Syntax

point = *Glob.GetFirstGlob*

See Also

GetNextGlob

Example

```
` build a list of all Globs in the MMF  
list1.Clear  
If Glob1.GetFirstGlob then  
    List1.Additem Glob1.GlobName  
    While Glob1.GetNextGlob  
        List1.additem Glob1.GlobName  
    Wend  
End If
```

GetNextGlob method

Description

Reads the next Glob in the Memory-mapped-file. Returns a Boolean indicating whether or not a valid Glob was found.

Syntax

Boolean = Glob.GetNextGlob

See Also

GetFirstGlob

Example

```
` build a list of all Globs in the MMF
list1.Clear
If Glob1.GetFirstGlob then
    List1.Additem Glob1.GlobName
    While Glob1.GetNextGlob
        List1.additem Glob1.GlobName
    Wend
End If
```

Link property

Description

The name of another glob that relates to this glob.

Syntax

strValue = Glob.Link
Glob.Link = strValue

Example

```
` position Glob2 to the position in Glob1.Link
Glob2.GlobName = Glob1.Link
```

Status property

Description

User-defined 16-bit value.

Syntax

Glob.Status = 16-bit-value

Remarks

Intended for use in conjunction with **Command** property for device control. By convention, the device 'owns' the **Status** property (can modify it), while the application must only read it.

See Also

Command

Example

```
Sub ReadMeasurement ( )
    On error goto Bad_Stuff
    GlobMeasurement.Value(0, 0) = AcquireSomeReading
    Exit Sub
Bad_Stuff:
    GlobMeasurement.Status = err
End Sub
```

Type

Description

User-defined 16-bit integer.. By convention *Type* contains values from 0 to 5 for integers that will be scaled into floating point numbers. In this case, the value is the number of decimal places implied in the integer.

Further convention uses *Type* = -1 to indicate a Label Glob, containing no data.

Syntax

Value = *Glob.Type*

Example

```
If Glob1.Type = -1 then
    Printer.Print Glob1.GlobName & "Is A Label"
End If
```

UOM property

Description

The name of a unit-of-measure label Glob to be associated with this Glob.

Syntax

```
strValue = Glob.UOM
Glob.UOM = strValue
```

Example

```
If Glob1.UOM = GlobPercent.GlobName Then
    ' Glob1 uses unit-of-measure from GlobPercent
End If
```

Value property

Description

This is one of the methods for accessing the data portion of a Glob from within a Visual Basic program.

Syntax

Glob.Value(<dim2>, <dim1>) = *newvalue* ' set the value into the Glob

Globvalue = *Glob.Value*(<dim2>, <dim1>) ' get the value

Remarks

If the array location results in a data element that is out of range, the new value

will not be written to the Glob. If attempting to read a data element with an invalid array location, **Value** will return a -1.

The return value is LONG (32-bit integer); if accessing data whose data elements are only 1 or 2 bytes long, the data values will be returned in the low order 8 or 16 bits of the 32-bit return value, as appropriate.

Example

```
Dim i as integer
\
\ fill a 512 byte Glob array of bytes with random values
\
Glob1.AddNew "WAVEFORM", 0, 1, 512, 1, 0, 0      \ dim1=512
For i = 0 to 511
    Glob1.Value( 0, i ) = rnd() * 255          \ save rnd byte
Next i
\
\ link with the "Dwell per cylinder" Glob and print values
\
Glob1.GlobName = "DWELL/CYL"                    \ link up
For i = 0 to NumberOfCylinders - 1              \ loop through
    Printer.Print Glob1.Value( 0, i )           \ access & print
Next i
```

EXTENDED GLOB METHODS AND PROPERTIES

The following methods and properties are new to the Glob control and are not available to the Blob control.

AutoSendNotify Property

Description

If this property is 'True' then any change in any of the 'Value' properties or in the 'Status' or 'Command' properties will cause the Change event to be fired.

Syntax

Glob1.AutoSendNotify = <True/False>

Remarks

When this property is set to False, any changes in the **value**, **status**, or **command** properties will not cause a **Change** event to be fired by the current Glob interface. Other glob interfaces that change the properties will still cause a Change event. To fire a Change event, use the **SendNotify** method. This method will send a Change event to all glob interfaces that are registered to be notified. This functionality allows an entire array to be updated without sending a change event for each element in the array. The event can be sent at the end of the update with the SendNotify method.

Examples

```
Glob1.AutoSendNotify = False
For x = 0 to 99
    Glob1.ValueSD(x) = data(x)
Next x
Glob1.SendNotify ID_VALUE, 0
Glob1.AutoSendNotify = True
```

aValue8, aValue16, aValue32 Properties

Description

Array versions of the Value8, Value16, and Value32 properties

Syntax

```
Glob1.aValue8(0) = my_byte      'write a byte value to the Glob
my_byte = Glob1.aValue16(5)    'read a byte value from the Glob
```

Remarks

This property accesses the data in the Glob's data area as a single dimension array of the appropriate type. Due to the limitations of the Automation types, aValue8() returns a short integer (2 bytes), aValue16() returns a short integer, and aValue32() returns a long integer (4 bytes).

See Also

Value8, Value16, Value32

Example

```
Dim i as integer
'
' fill a 512 byte Glob array of bytes with random values
'
Glob1.AddNew "WAVEFORM", 0, 1, 512, 1, 0, 0      ' dim1=512
For i = 0 to 511
    Glob1.aValue8( i ) = rnd() * 255          ' save rnd byte
Next i
'
' link with the "Dwell per cylinder" Glob and print values
'
Glob1.GlobName = "DWELL/CYL"                    ' link up
For i = 0 to NumberOfCylinders - 1              ' loop through
    Printer.Print Glob1.aValue8( i )            ' access & print
Next i
```

Change Event

Description

This event is triggered if a Glob has registered for notification and the data in the Globs data area has been changed.

4/24/98 update:

This event will also fire if the status or command fields have been changed. Two parameters are passed in with the event as well. The first identifies the property that changed. The second is a data value that is user defined or in the case of a status or command change, the new values of these properties.

Syntax

```
Sub Glob1_Change(ByVal PropID as integer, ByVal Value as integer,
                ByVal SendID as long)
    'TO-DO: Put code here to handle change event
Exit sub
```

Remarks

The following PropID values are predefined and will be passed to the Change event if the AutoSendNotify property is set to True.

ID_UNKNOWN	=	0
ID_VALUE_CHANGED	=	1
ID_STATUS_CHANGED	=	2
ID_COMMAND_CHANGED	=	3

The SendID parameter contains the GlobIndex of the Glob that changed. This parameter can be used along with the NotifyOnChange property to have one glob interface monitor many Globs for changes.

Example

```
Sub Glob1_Change(ByVal PropID as integer, ByVal Value as
integer, ByVal SendID as long)

    'identify which property changed
    Select Case PropID
        Case ID_VALUE_CHANGED:
            'process value change
        Case ID_STATUS_CHANGED:
            'process status change
        Case ID_COMMAND_CHANGED:
            'process command change
        Case ID_UNKNOWN:
            'process something.
    End Select
Exit sub
```


FormatMMF Method

Description

This method allows the MMF to be reformatted.

Syntax

Glob1.FormatMMF(<integer>)

Remarks

The integer value represents the maximum number of Notification registrations that the newly formatted MMF can handle. The default MMF can handle 256 registrations. This method call will destroy all data currently in the MMF file. This call should be made before adding any Globs with the AddNew function.

It should be noted that the fewer notification that a particular MMF will have to handle, the smaller the MMF file will be. Large numbers of notification registrations may slow the performance of the application using the MMF.

Example

```
Glob1.FormatMMF(1024)
.
.
.
Glob1.AddNew "AMPS",0,0,0,0,0,0
```

FullPath Property

Description

Returns fully qualified path to the MMF being used.

Syntax

Path = Glob1.FullPath

Remarks

Read only. Returns a string value. This property is usefull to determine if the Glob is connected to the proper MMF.

See Also

FileName Property

GlobSize Property

Description

This property returns the total size in bytes of the Glob. This includes the Glob header area, data area, and the notify map.

Syntax

N = Glob1.GlobSize

Remarks

Each Glob in the MMF can be different sizes. This property can be used to chain through the globs in the MMF without using the GetNextGlob function. The next glob in the file has an index equal to the current GlobIndex + GlobSize.

Example

```
Dim NewIndex as long

'Get the next glob in the MMF.
NewIndex = Glob1.GlobIndex + Glob1.GlobSize
Glob1.GlobIndex = NewIndex
```

IndexOf Property

Description

Returns the index into the glob MMF of a named glob.

Syntax

My_Index = Glob1.IndexOf(<string>)

Remarks

This method can be used to retrieve Glob Indexes from a Glob Interface without altering which Glob the Interface points to.

Example

```
'Find the index of the UOM_AMPS glob

Uom_Amps_Index = Glob1.IndexOf("UOM_AMPS")

'This call does not change which glob Glob1 is pointing at.
'This preserves any notification settings in Glob1. The
'index could have been found by setting Glob1.GlobName to
'"UOM_AMPS" and then reading the Glob1.GlobIndex property
'but any Notification settings would have been lost.
```

Insert Method

Description

Allows a value to be inserted into the middle of an array of values, moving values after the insert point up one position.

Syntax

`Glob1.Insert(<value>,<position>)`

<value> is a long integer

<position> is a zero based index into the array

Remarks

The insert method relies on the elsize property to determine the location to insert into the Glob's memory area. Values stored in the last index location of the array are rolled off the end and lost. <value> is a long integer and input should be cast as a long regardless of the actual data type. The Glob will use the elsize property to determine the actual data type and store it accordingly.

Example

```
'keep a history of the last 10 access times
Sub ReadFile()
    Glob1.Insert(Now,0)           'store date/time inf.
    'perform data access functions
    :
    exit sub
```

nHandles Property

Description

This property returns the number of notification handles that this particular MMF has been formatted to handle.

Syntax

`N = Glob1.nHandles`

Remarks

This property is read only.

Example

```
Dim I as integer

For I = 0 to Glob1.nHandles - 1
    List1.AddItem Glob1.NotifyHandle(I)
Next I
```

nNotifyMaps Property

Description

This property returns the number of Notification maps that each Glob contains.

Syntax

`N = Glob1.nNotifyMaps`

Remarks

This property is read only.

Example

```
Dim I as integer

For I = 0 to Glob1.nNotifyMaps - 1
    List1.AddItem Glob1.NotifyMap(I)
Next I
```

Notify Property

Description

Boolean value. If set to **True** then the control will raise a **Change** event if any portion of the data area is modified.

4/24/98 update:

The **Change** event will also fire if the **Status** or **Command** properties are altered.

Syntax

`Glob1.Notify = <True/False>`
`Is_Notify_On = Glob1.Notify`

Remarks

Each MMF file is capable of supporting a fixed number of controls to be notified of a data change. The limit is global in scope, meaning that only a fixed number of controls across the entire system can be registered for notification for a particular MMF file. If an attempt is made to set the **Notify** property of a control to **True** and the register is full, the property will not change, it will retain a **False** value.

If the MMF location that the Glob is currently "looking at" is changed, the **Notify** property will automatically remove the Glob's handle from the **NotifyHandle** list and set itself to **False**. This could happen if the **GlobName**, **GlobIndex**, **FileName**, or **GlobPointer** properties are changed.

If an application containing Globbs that are registered for notification exits in a normal manner, the Globbs will automatically remove their handles from the NotifyHandle list stored in the MMF. If the application terminates abnormally, there is a possibility that invalid handles will be left in the list. In order to handle this, if there are no Globbs currently connected to a specific MMF, the first Globb to connect will clear the NotifyHandle list. Also, during the notification process, if an invalid window handle is encountered, the invalid entry is automatically removed from the list. Exiting all applications that are using a particular MMF and then restarting them is guaranteed to clear out all invalid entries in the list.

4/24/98 update:

The Notify property is now read-only at design time.

Examples

```
'set up a app to be notified of data changes in a glob
```

```
Glob1.Notify = True  
if (Glob1.Notify = False) then  
    msgbox "unable to register Glob1 for notification"  
end if
```

NotifyHandle Property

Description

Read only list of all the window handles currently registered for notification in a particular MMF.

Syntax

```
a_windows_handle = Glob1.NotifyHandle(<0-??>)
```

Remarks

The NotifyHandle list is global to ALL Globbs pointing at the same MMF, even Globbs in separate processes. The values returned are the window handles of the individual controls that are registered for notification. A zero entry indicates an unused (available) location in the list. This property is included primarily as a debug tool for developers.

Examples

```
'check to see if a particular control is registered.  
  
Glob1.Notify = True  
NotifyOK = False  
For I = 0 to 31  
    if Glob1.hWnd = Glob1.NotifyHandle(I) then  
        NotifyOK = true  
    end if  
next I  
  
if NotifyOK = False then  
    msgbox "Did not find Glob1's handle in the list"  
end if
```

NotifyMap Property

Description

An array of 32 bit (long integer) values representing a bit map of the NotifyHandle list entries to notify when the Glob value changes. The NotifyMap property is read-only.

Syntax

`bit_map = Glob1.NotifyMap(x)`

Remarks

by examining the bit pattern stored in this property, the exact NotifyHandle entries that will be notified of a change can be determined. If bit 0 is set to 1 then NotifyHandle entry 0 will be notified of a change if any of this Glob's value properties are changed. Bits 0-31 map directly to NotifyHandle entries 0-31 in the MMF. This property was included to aid in debugging applications. When used in conjunction with the NotifyHandle list, the programmer can monitor exactly which Glob's are getting sent the change message when a value is changed.

Example

```
'Display a list of handles that will be notified if the
'value of this glob is changed.

my_Map = Glob1.NotifyMap(0)
For x = 0 to 31
    bitmask = 2^x           'a left bit shift operation
                             'would be better here.
    if (my_Map And bitmask) then
        ListBox.AddItem Glob1.NotifyHandle(x)
    end if
next x
```

NotifyOnChange Property

Description

This property allows a single GlobInterface module to monitor many 'Globs' in the MMF for changes.

Syntax

`Glob1.NotifyOnChange(<GlobName>) = <True/False>`
`IsNotified = Glob1.NotifyOnChange(<GlobName>)`

Remarks

NotifyOnChange allows a single Glob interface to receive notification messages from multiple Glob's in the MMF. Note that if the Notify Property is set to 'False' that ALL notification registrations for this Glob Interface are deleted. (see example)

Example

```
'Register Glob1 to receive notification from 4 Glob's in the
'MMF.
```

```
Glob1.NotifyOnChange("AMPS") = true  
Glob1.NotifyOnChange("VOLTS") = true  
Glob1.NotifyOnChange("CO2") = true  
Glob1.NotifyOnChange("NOX") = true
```

'The following statement clears all notifications for Glob1

```
Glob1.Notify = false
```

'The above statement is the same as the following code.

```
Glob1.NotifyOnChange("AMPS") = false  
Glob1.NotifyOnChange("VOLTS") = false  
Glob1.NotifyOnChange("CO2") = false  
Glob1.NotifyOnChange("NOX") = false
```

RefCount Property

Description

This property exposes the MMF's reference count variable.

Syntax

N = Glob1.RefCount

Remarks

There are two cases where the Reference Count of the Globbs interfacing with a particular MMF are important. The AddNew method will fail if the RefCount is greater than 1 and the AddNew results in the MMF being expanded. Also, the FormatMMF method must be used with care if the RefCount is greater than 1. If another application is using the MMF when it is reformatted, the results will be unpredictable and the application may crash.

SendNotify Method

Description

This method is used to manual send a notification message to all glob interfaces that are registered to receive one.

Syntax

Glob1.SendNotify(<PROP_ID>, <VALUE>)

Remarks

The <PROP_ID> parameter is used to pass a property id number to the change event of the receiving glob interfaces. The following ID's are pre-defined.

ID_UNKNOWN	=	0
ID_VALUE_CHANGED	=	1
ID_STATUS_CHANGED	=	2
ID_COMMAND_CHANGED	=	3

ID values above 3 are available for custom ID's.

The <VALUE> parameter is used to pass the new value of the changed property. In the case of a data value, this is usually 0 since the data may be stored in any format in the glob. For the status and command properties this value should be the new value of the property.

Example

See the AutoSendNotify property.

StrValue Property

Description

This property returns or sets a string value into a Glob.

Syntax

```
Glob1.StrValue = <string>  
my_string = Glob1.StrValue  
<string> can be any valid string literal or a string variable.
```

Remarks

Globs store a string as a NULL terminated array of bytes in its data area. Care must be taken that the byte length of a string is not larger than the DataSize of the Glob. If a string is too large to fit then it will be truncated to fit into the data area of the Glob.

Visual Basic vs. Visual C++ (MFC)

In Visual Basic, this property works exactly as expected, VB handles all the conversions to and from C style strings transparently. In Visual C++, when setting the StrValue property, <string> is expected to be of the type LPCTSTR or a CString object. When reading StrValue, the return type is a BSTR object. The safest way to access the StrValue property from within a Visual C++ program is to use a CString object for both the assignment and retrieval of the value. The CString objects assignment operator (=) is overloaded to handle the conversions between LPCTSTR types and BSTR objects. String literal can be used to set the value in the Glob. The _T(<string_literal>) macro should be used to insure that the value is stored in the proper format (Unicode or Double Byte characters)

Examples

Visual Basic

```
Dim my_string as string

Glob1.StrValue = "This is a Glob"
Glob1.StrValue = my_string

my_string = Glob1.StrValue
```

Visual C++ (MFC)

```
CString          my_string;

//using the _T macro to assign a string literal
Glob1.SetStrValue(_T("This is a Glob"));

//using a CString object to access the property
Glob1.SetStrValue(my_string);
my_string = Glob1.GetStrValue();
```

Value8, Value16, Value32 Properties

Description

Allows reading and setting of values to the Glob's data area.
the Value32 property is the default value property for the Glob

Syntax

```
Glob.Value8 = new_byte_value      ' set value of the data area

byte_value = Glob.Value8          ' retrieve the value
```

Remarks

Value8, Value16, and Value32 return a single byte value, a 2 byte value, and a 4 byte value respectively. If a value is written to a Glob whose data area is smaller than the byte size of the value, nothing will be written to the Glob. Value32 is the default value.

Example

```
Dim my_byte as byte
my_byte = Cbyte(Glob1.Value8)

OR

Dim my_byte as integer
my_byte = Glob1.Value8

OR

'using the default property
my_long = Glob1
Glob1 = my_long
```

ValueSD property

Description

This is one of the methods for accessing the data portion of a Glob from within a Visual Basic program.

Syntax

Glob.ValueSD(<index>) = newvalue ' set the value into the Glob

Globvalue = Glob.ValueSD(<index>) ' get the value

Remarks

This property is a single dimensional version of the **Value** property and behaves in exactly the same manner. The **ValueSD** property takes only one index parameter instead of two. The memory associated with the Glob is treated as a single dimensional array for access purposes.

If the array specifiers result in a data element that is out of range, the new value will not be written to the Glob. If attempting to read a data element with invalid array specifiers, **Value** will return a -1.

The return value is LONG (32-bit integer); if accessing data whose data elements are only 1 or 2 bytes long, the data values will be returned in the low order 8 or 16 bits of the 32-bit return value, as appropriate.

See Also

Value property

Example

```
Dim i as integer
'
' fill a 512 byte Glob array of bytes with random values
'
Glob1.AddNew "WAVEFORM", 0, 1, 512, 1, 0, 0 ' dim1=512
For i = 0 to 511
    Glob1.ValueSD( i ) = rnd() * 255 ' save rnd byte
Next i
'
' link with the "Dwell per cylinder" Glob and print values
'
Glob1.GlobName = "DWELL/CYL" ' link up
For i = 0 to NumberOfCylinders - 1 ' loop through
    Printer.Print Glob1.ValueSD( i ) ' access & print
Next i
```

Version Property

Description

Read only property that returns a string containing the version ID of the Glob interface.

Syntax

version = Glob1.Version

Remarks

Since there have been several versions of this control produced, this property was added to help developers tell which version they had installed on their machine. The following conventions will be used in future releases:

Version ID = 1.2x

Primary release number = 1: This number will increase if there are substantial feature changes in the Glob Interface.

Secondary release number = 2: This number will increase if bug fixes and minor feature updates will invalidate the current MMF structure.

Release version letter = x: This will increment if bug fixes and feature updates Do not require any MMFs being used to be rebuilt.

Stock Properties

Visual Basic and Visual C++ automatically provide several stock properties for any custom control. Documentation for these properties can be found in the on-line help system for either Visual Basic or Visual C++

Some standard stock properties

hWnd, Top, Left, Height, Width, Index, Icon...

Update History

Version 1.2a (May 4, 1998)

- 1) Fixed a problem with Deleting and then Adding new globs to the MMF.
- 2) Link and UOM properties are now strings instead of indexes.
- 3) added the IndexOf method call to retrieve the index of a named Glob.
- 4) Added the version property to make it easier to keep track of which version is being used.

Version 1.2b (May 5, 1998)

- 1) Added SendID to the Change event.
- 2) Some internal modifications to the way messages are sent from Glob to Glob.
- 3) Optimized the Notification process if there are no notifications registered.
- 4) Added the NotifyOnChange property.

Version 1.2c (May 6, 1998)

- 1) Fixed a bug in the NotifyOnChange property. Users can now set up a Glob with no name to receive notifications from other Globs.
- 2) Fixed a problem with creating a new file if the specified MMF does not exist.

Version 1.2d (??)

- 1) Fixed a crash in Windows95 if the MMF file did not exist.
- 2) Windows 95 no longer crashed when the MMF is expanded.

Version 1.2e (June 16, 1998)

- 1) Removed Glob interfaces from the control tab list.
- 2) Globs no longer display a "Property Page" when clicked on during runtime.

Required Files

The following files are required on the system before Glob.ocx can be successfully registered and used.

MFC42.DLL *
OLEPRO32.DLL *
MSVCRT.DLL *
GLOB.OCX

Note:

The files marked with *'s will be present on systems that have Internet Explorer 3.0 or Microsoft Visual C++ with MFC version 4.2 installed. If these files are not found in the system or system32 directory then they must be installed and registered before Glob.ocx can be registered.

APPENDIX B

```
// Copyright 1998, 1999 SPX Corporation
// Glob.cpp: Implementation of CGlobApp and DLL registration.
```

```
#include "stdafx.h"
#include "Glob.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

CGlobApp NEAR theApp:

```
const GUID CDECL BASED_CODE _tlid =
    { 0x5f20d2d3, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };
const WORD _wVerMajor = 1;
const WORD _wVerMinor = 0;
```

```
////////////////////////////////////
//CGlobApp::InitInstance - DLL initialization
```

```
BOOL CGlobApp::InitInstance()
{
    BOOL bInit = COleControlModule::InitInstance();

    if (bInit)
        // TODO: Add your own module initialization code here.

    return bInit;
}
```

```
////////////////////////////////////
// CGlobApp::ExitInstance - DLL termination
```

```
int CGlobApp::ExitInstance()
{
    // TODO: Add your own module termination code here.

    return COleControlModule::ExitInstance();
}
```

```
////////////////////////////////////
//DllRegisterServer - Adds entries to the system registry
```

```
STDAPI DllRegisterServer(void)
```

```

{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleRegisterTypeLib(AfxGetInstanceHandle(), _tlid))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (IColeObjectFactoryEx::UpdateRegistryAll(TRUE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

```

```

////////////////////////////////////
// DllUnregisterServer - Removes entries from the system registry

```

```

STDAPI DllUnregisterServer(void)
{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleUnregisterTypeLib(_tlid, _wVerMajor, _wVerMinor))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (IColeObjectFactoryEx::UpdateRegistryAll(FALSE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

```

: Copyright 1998, 1999 SPX Corporation
: Glob.def : Declares the module parameters.

LIBRARY "GLOB.OCX"

EXPORTS

DllCanUnloadNow @1 PRIVATE
DllGetClassObject @2 PRIVATE
DllRegisterServer @3 PRIVATE
DllUnregisterServer @4 PRIVATE

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211

```
// Copyright 1998, 1999 SPX Corporation
// Glob.odl : type library source for ActiveX Control project.

// This file will be processed by the Make Type Library (mktypelib) tool to
// produce the type library (Glob.tlb) that will become a resource in
// Glob.ocx.
```

```
#include <olectl.h>
#include <idispatch.h>
```

```
[ uuid(5F20B2D3-788C-11D1-9A9B-020701045A6B), version(1.0),
  helpfile("Glob.hlp"),
  helpstring("Glob MMF Interface"),
  control ]
```

```
library GLOBLib
```

```
{
  importlib(STDOLE_TLB);
  importlib(STDTYPE_TLB);
```

```
// Primary dispatch interface for CGlobCtrl
```

```
[ uuid(5F20B2D4-788C-11D1-9A9B-020701045A6B),
  helpstring("Dispatch interface for Glob Control"), hidden ]
dispatchinterface _DGlob
```

```
properties:
```

```
    // NOTE - ClassWizard will maintain property information here.
```

```
    // Use extreme caution when editing this section.
```

```
    //[{AFX_ODL_PROP(CGlobCtrl)
```

```
    [id(DISPID_HWND)] OLE_HANDLE hWnd;
```

```
    [id(1)] long GlobIndex;
```

```
    [id(2)] long Dim1Size;
```

```
    [id(3)] long Dim2Size;
```

```
    [id(4)] long ElementSize;
```

```
    [id(5)] long Type;
```

```
    [id(6)] long Extra;
```

```
    [id(7)] long DataSize;
```

```
    [id(8)] BSTR GlobName;
```

```
    [id(9)] long FileSize;
```

```
    [id(10)] BSTR FileName;
```

```
    [id(11)] long Status;
```

```
    [id(12)] long Command;
```

```
    [id(13)] long GlobPtr;
```

```
    [id(14)] long DataPtr;
```

```
    [id(15)] long AvailSize;
```

```
    [id(16)] boolean ReadOnlyMMF;
```

```
    [id(17)] boolean Notify;
```

```
    [id(18)] short Value8;
```

```
    [id(19)] long Value32;
```

```
    [id(20)] short Value16;
```

```
    [id(21)] BSTR StrValue;
```

```
    [id(22)] BSTR FullPath;
```



```

[id(23)] boolean AutoSendNotify;
[id(24)] long nHandles;
[id(25)] long nNotifyMaps;
[id(26)] long GlobSize;
[id(27)] long RefCount;
[id(28)] BSTR Version;
[id(29)] BSTR UOM;
[id(30)] BSTR Link;
//}}AFX_ODL_PROP

```

methods:

```

// NOTE - ClassWizard will maintain method information here.
// Use extreme caution when editing this section.
//{{AFX_ODL_METHOD(CGlobCtrl)
[id(43), propget] long Value(long Dim2, long Dim1);
[id(43), propput] void Value(long Dim2, long Dim1, long nNewValue);
[id(31)] long CloseMMF();
[id(32)] long AddNew(BSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra);
[id(33)] long AddNewEx(BSTR GlobName, BSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra);
[id(34)] boolean GetFirstGlob();
[id(35)] boolean GetNextGlob();
[id(36)] void Erase();
[id(37)] void EraseMMF();
[id(44), propget] long NotifyHandle(short index);
[id(44), propput] void NotifyHandle(short index, long nNewValue);
[id(45), propget] short aValue8(long index);
[id(45), propput] void aValue8(long index, short nNewValue);
[id(46), propget] long aValue32(long index);
[id(46), propput] void aValue32(long index, long nNewValue);
[id(47), propget] short aValue16(long index);
[id(47), propput] void aValue16(long index, short nNewValue);
[id(48), propget] long ValueSD(long n);
[id(48), propput] void ValueSD(long n, long nNewValue);
[id(38)] void Insert(long value, long index);
[id(39)] long ResizeMMF(long NewSize);
[id(40)] void SendNotify(long NotifyID, long Value);
[id(49), propget] long NotifyMap(long index);
[id(49), propput] void NotifyMap(long index, long nNewValue);
[id(41)] boolean FormatMMF(long NotifyLimit);
[id(42)] long IndexOf(BSTR GlobName);
[id(50), propget] boolean NotifyOnChange(BSTR GlobName);
[id(50), propput] void NotifyOnChange(BSTR GlobName, boolean bNewValue);
//}}AFX_ODL_METHOD

[id(DISPID_ABOUTBOX)] void AboutBox();
};

```

// Event dispatch interface for CGlobCtrl

[uuid(5F20D2D5-788C-11D1-9A9B-020701045A6B),

```

helpstring("Event interface for Glob Control") ]
dispinterface _DGlobEvents
{
    properties:
        // Event interface has no properties

    methods:
        // NOTE - ClassWizard will maintain event information here.
        // Use extreme caution when editing this section.
        //{{AFX_ODL_EVENT(CGlobCtrl)
        [id(1)] void Change(short PropID, short Value, long SendID);
        //}}AFX_ODL_EVENT
};

```

```

// Class information for CGlobCtrl

```

```

[ uuid(5F20D2D6-788C-11D1-9A9B-020701045A6B), licensed,
  helpstring("Glob Control"), control ]

```

```

coclass Glob

```

```

{
    [default] dispinterface _DGlob;
    [default, source] dispinterface _DGlobEvents;
}

```

```

//{{AFX_APPEND_ODL}}
//}}AFX_APPEND_ODL}}

```

```

};

```

```
// Copyright 1998, 1999 SPX Corporation
// GlobCtrl.cpp : Implementation of the CGlobCtrl ActiveX Control class.
```

```
#include "stdafx.h"
#include "Glob.h"
#include "GlobCtrl.h"
#include "GlobPpg.h"
#include "sys/types.h" // for file status buffer _stat
#include "sys/stat.h" // for _fstat file status call
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
#define MIN_FILESIZE 8192
#define LOG_ERRORS FALSE
```

```
HANDLE myhWnd;
```

```
HANDLE ADhInstance;
unsigned char bResFlag;
HANDLE *iaModules;
int *iaGlobal;
BSTR saGlobal;
CString refTemp;
```

```
//BOOL LineDebug = false;
```

```
LPCTSTR m_Message = "msgGlobChange";
//long BitList[32];
```

```
// HELPER FUNCTION PROTOTYPES -RK
```

```
CString GetName(CString);
long Power(int);
void LogErrorString(CString errstr);
```

```
IMPLEMENT_DYNCREATE(CGlobCtrl, COleControl)
```

```
////////////////////////////////////
//Register for an external windows message
```

```
UINT USER_VALUECHANGED = RegisterWindowMessage(m_Message);
```

```
////////////////////////////////////
// Message map
```

```
BEGIN_MESSAGE_MAP(CGlobCtrl, COleControl)
//{{AFX_MSG_MAP(CGlobCtrl)
//}}AFX_MSG_MAP
```

////////////////////////////////////

```
BEGIN_DISPATCH_MAP(CGlobCtrl, COleControl)
```

```
DISP_PROPERTY_EX(CGlobCtrl, "GlobIndex", GetGlobIndex, SetGlobIndex, VT_I4)
```

```
DISP_PROPERTY_EX(CGlobCtrl, "Dim2Size", GetDim2Size, SetDim2Size, VT_I4)
```

```
DISP_PROPERTY_EX(CGlobCtrl, "Type", GetType, SetType, VT_I4)
```

```
NTSP_PROPERTY_EX(CGlobCtrl, Extra, GetExtra, SetExtra, 1, 1);
NTSP_PROPERTY_EX(CGlobCtrl, "DataSize", GetDataSize, SetDataSize,
```

```
ULSP_PROPERTY_EX(CGlobCtrl, GlobName, GetGlobName, SetGlobName, VT_I4)  
21 LSP_PROPERTY_EX(CGlobCtrl, "FileSize", GetFileSize, SetFileSize, VT_I4)
```

DISP_PROPERTY_EX(CGlobeCtrl, "FileName", GetFileName, SetFileName, VT_BSTR)
DISP_PROPERTY_EX(CGlobeCtrl, "Status", GetStatus, SetStatus, VT_BOOL)

DISP_PROPERTY_EX(CGlobCtrl, "Command", GetCommand, SetCommand, V1_14)

```
DISP_PROPERTY_EX(CGlobCtrl, "DataPtr", GetDataPtr, SetDataPtr, VT_I4)
```

```
DISP_PROPERTY_EX(CGlobCtrl, "ReadOnlyMMF", GetReadOnlyMMF, SetReadOnlyMMF, VT_BOOL)
```

```
DISP_PROPERTY_EX(CGlobCtrl, "Value8", GetByteValue, SetByteValue, VT_I2)
```

```
DTSP_PROPERTY_EX(CGlobCtrl, "Value16", GetIValue, SetIValue, VT_I2)
```

DISP_PROPERTY_EX(CGlobeCtrl, "FullPath", GetFullPath, SetFullPath, VT_BSTR)

DISP_PROPERTY_EX(CGlobalCtrl, "AutoSendNotify", GetAutoSendNotify, SetAutoSendNotify, VT_BOOL, 0)
DISP_PROPERTY_EX(CGlobalCtrl, "NHandles", GetNHandles, SetNHandles, VT_I4, 0)

DISP_PROPERTY_EX(CGlobCtrl, "nNotifyMaps", GetNNotifyMaps, SetNNotifyMaps)

DISP_PROPERTY_EX(CSlobCtrl, "RefCount", GetRefCount, SetRefCount, VT_I4)

DISP_PROPERTY_EX(CGlobCtrl, "UOM", GetUOM, SetUOM, VT_BSTR)

```
DISP_FUNCTION(CGlobCtrl, "CloseMMF", MMFClose, VT_I4, VTS_NONE)
```

DISP_FUNCTION(CGlobCtrl, "AddNewEx", MMFAddGlobEx, VT_I4, VTS_BSTR VTS_BSTR VTS_I4 VTS_I4 VTS_I4 VTS_I4)

DISP_FUNCTION(CGlobCtrl, "GetNextGlob", GetNextGlob, VT_BOOL, VTS_NONE)

DISP_FUNCTION(CGlibCgl, "Erase", Erase, VT_EMPTY, VTS_NONE)
DISP_FUNCTION(CGlibCgl, "EraseName", EraseName, VT_EMPTY, VTS_NONE)DISP_FUNCTION(CGlobCtrl, "Insert", Insert, VI_EMPTY, VIS_14 VIS_15
DISP_FUNCTION(CGlobCtrl, "Delete", Delete, NAME, VI_71 VI_72 VI_73 VI_74)

DISP_FUNCTION(CGlobCtrl, "SendNotify", SendNotifyX, VT_EMPTY, VIS_12)

DISP_FUNCTION(CGlobCtrl, "IndexOf", IndexOf, VT_I4, VTS_BSTR)

```

DISP_PROPERTY_PARAM(CGlobCtrl, "Value", GetValue, SetValue, VT_I4, VTS_I4 VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyHandle", GetNotifyHandle, SetNotifyHandle, VT_I4, VTS_I2)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue8", GetAbValue, SetAbValue, VT_I2, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue32", GetAiValue, SetAiValue, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue16", GetAiValue, SetAiValue, VT_I2, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "ValueSD", GetValueSD, SetValueSD, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyMap", GetNotifyList, SetNotifyList, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyOnChange", GetNotifyOnChange, SetNotifyOnChange, VT_BOOL, VTS_BSTR)
DISP_DEFVALUE(CGlobCtrl, "Value32")
DISP_STOCKPROP_HWND()
//}}AFX_DISPATCH_MAP
DISP_FUNCTION_ID(CGlobCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY, VTS_NONE)
END_DISPATCH_MAP()

```

```

/////////////////////////////////////////////////////////////////
// Event map

```

```

BEGIN_EVENT_MAP(CGlobCtrl, COleControl)
//{{AFX_EVENT_MAP(CGlobCtrl)
EVENT_CUSTOM("Change", FireChange, VTS_I2 VTS_I2 VTS_I4)
//}}AFX_EVENT_MAP
END_EVENT_MAP()

```

```

/////////////////////////////////////////////////////////////////
// Property pages

```

```

// TODO: Add more property pages as needed. Remember to increase the count!
BEGIN_PROPPAGEIDS(CGlobCtrl, 1)
PROPPAGEID(CGlobPropPage::guid)
END_PROPPAGEIDS(CGlobCtrl)

```

```

/////////////////////////////////////////////////////////////////
// Initialize class factory and guid

```

```

IMPLEMENT_OLECREATE_EX(CGlobCtrl, "GLOB.GlobCtrl.1",
    0x5f20d2d6, 0x788c, 0x11d1, 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b)

```

```

/////////////////////////////////////////////////////////////////
// Type library ID and version

```

```

IMPLEMENT_OLETYPELIB(CGlobCtrl, _tlid, _wVerMajor, _wVerMinor)

```

```

/////////////////////////////////////////////////////////////////
// Interface IDs

```

```

const IID BASED_CODE IID_DGlob =
    { 0x5f20d2d4, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };

```

```
const IID BASED_CODE IID_DGlobEvents =
    { 0x5f20d2d5, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };
```

```
////////////////////////////////////
// Control type information
```

```
static const DWORD BASED_CODE _dwGlobOleMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;
```

```
IMPLEMENT_OLECTLTYPE(CGlobCtrl, IDS_GLOB, _dwGlobOleMisc)
```

```
////////////////////////////////////
```

```
// CGlobCtrl::CGlobCtrlFactory::UpdateRegistry -
// Adds or removes system registry entries for CGlobCtrl
```

```
BOOL CGlobCtrl::CGlobCtrlFactory::UpdateRegistry(BOOL bRegister)
```

```
{
    // TODO: Verify that your control follows apartment-model threading rules.
    // Refer to MFC TechNote 64 for more information.
    // If your control does not conform to the apartment-model rules, then
    // you must modify the code below, changing the 6th parameter from
    // afxRegApartmentThreading to 0.
```

```
    if (bRegister)
        return AfxOleRegisterControlClass(
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_GLOB,
            IDB_GLOB,
            afxRegApartmentThreading,
            _dwGlobOleMisc,
            _tlid,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}
```

```
////////////////////////////////////
```

```
// Licensing strings
```

```
static const TCHAR BASED_CODE _szLicFileName[] = _T("Glob.lic");
```

```
static const WCHAR BASED_CODE _szLicString[] =
    L"Copyright (c) 1999 SPX";
```

```
////////////////////////////////////
```

```
// CGlobCtrl::CGlobCtrlFactory::VerifyUserLicense -
```

```
// Checks for existence of a user license
```

```
BOOL CGlobCtrl::CGlobCtrlFactory::VerifyUserLicense()
```

```
{  
    return AfxVerifyLicFile(AfxGetInstanceHandle(), _szLicFileName,  
        _szLicString);  
}
```

```
////////////////////////////////////
```

```
// CGlobCtrl::CGlobCtrlFactory::GetLicenseKey -
```

```
// Returns a runtime licensing key
```

```
BOOL CGlobCtrl::CGlobCtrlFactory::GetLicenseKey(DWORD dwReserved,
```

```
BSTR FAR* pbstrKey)
```

```
{  
    if (pbstrKey == NULL)
```

```
    return FALSE;
```

```
    *pbstrKey = SysAllocString(_szLicString);  
    return (*pbstrKey != NULL);  
}
```

```
////////////////////////////////////
```

```
// CGlobCtrl::CGlobCtrl - Constructor
```

```
CGlobCtrl::CGlobCtrl()
```

```
{
```

```
    InitializeIIDs(&IID_DGlob, &IID_DGlobEvents);
```

```
    // TODO: Initialize your control's instance data here.
```

```
    SetInitialSize( 32, 32 ); // Force to have a certain size at startup
```

```
    lpLast = NULL;
```

```
    lpView = NULL;
```

```
    s_hFileMap = NULL;
```

```
    hFileMapT = NULL;
```

```
    f = NULL;
```

```
    GlobLock = NULL;
```

```
    MMFLock = NULL;
```

```
    m_FileName = "C:\\GLOBMMF";
```

```
    m_FileSize = MIN_FILESIZE;
```

```
    GlobPtr = 0;
```

```
    m_Notify = false;
```

```
}
```

```

////////////////////////////////////
// CGlobCtrl::~CGlobCtrl - Destructor

```

```

CGlobCtrl::~CGlobCtrl()

```

```

{
    // TODO: Cleanup your control's instance data here.

```

```

    RemoveNotify(GlobPtr, GetSafeHwnd());

```

```

    if (lpView) {

```

```

        lpView->RefCount--;

```

```

        UnmapViewOfFile((LPVOID) lpView);

```

```

        GlobPtr = NULL;

```

```

        lpView=NULL;

```

```

        lpLast=NULL;

```

```

        CloseHandle(s_hFileMap);

```

```

        CloseHandle(f);

```

```

    }

```

```

    if (GlobLock) delete GlobLock;

```

```

    GlobLock = NULL;

```

```

    if (MMFLock) delete MMFLock;

```

```

    MMFLock = NULL;

```

```

}

```

```

////////////////////////////////////
// CGlobCtrl::OnDraw - Drawing function

```

```

void CGlobCtrl::OnDraw(

```

```

    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)

```

```

{

```

```

    // TODO: Replace the following code with your own drawing code.

```

```

    CRect r;

```

```

    CPictureHolder pict;

```

```

    if (!AmbientUserMode()) {

```

```

        r = rcBounds;

```

```

        r.right = r.left + 31;

```

```

        r.bottom = r.top + 31;

```

```

        pict.CreateFromBitmap(IDB_GLOB);

```

```

        pict.Render(pdc, r, r);

```

```

        SetControlSize(32, 32);

```

```

    } else {

```

```

        ShowWindow(SW_HIDE);

```

```

    }

```

```

}

```

```

////////////////////////////////////
// CGlobCtrl::DoPropExchange - Persistence support

```

```

void CGlobCtrl::DoPropExchange(CPropExchange* pPX)

```



```

{
    CString strResult;
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);
    long ret;

    // TODO: Call PX_ functions for each persistent custom property.
    {
        // Make FileName property Persistent
        PX_String(pPX, _T("FileName"), m_FileName, "C:\\GLOBMMF");
        SetFileName(m_FileName);

        // make GlobName persistent
        PX_String (pPX, _T("GlobName"), m_GlobName, "");
        SetGlobName( m_GlobName ); // look up the Glob for this name, should relookup

        PX_Bool(pPX, _T("AutoNotify"), m_AutoNotify, true);
    }

    ret = SetVisible();
}

```

```

////////////////////////////////////
// CGlobCtrl::OnResetState - Reset control to default state

```

```

void CGlobCtrl::OnResetState()
{
    COleControl::OnResetState(); // Resets defaults found in DoPropExchange

    // TODO: Reset any other control state here.
}

```

```

////////////////////////////////////
// CGlobCtrl::AboutBox - Display an "About" box to the user

```

```

void CGlobCtrl::AboutBox()
{
    CDialog dlgAbout(IDD_ABOUTBOX_GLOB);
    dlgAbout.DoModal();
}

```

```

////////////////////////////////////
// CGlobCtrl message handlers

```

```

long CGlobCtrl::SetGlobIndex()
{

```

```

// TODO: Add your property handler here
if (GlobPtr)
    return (int)(GlobPtr)-(int)lpView;
return -1;
}

void CGlobCtrl::SetGlobIndex(long nNewValue)
{
    // TODO: Add your property handler here
    if ((nNewValue >= lpView->FirstGlob) && (nNewValue <= lpView->NextAvail)) // in range?...
    {
        RemoveNotify(GlobPtr, GetSafeHwnd());
        GlobPtr = (tGlob *)((int)lpView + nNewValue); // hope caller knows what he's doing
        if (GlobPtr)
        {
            datasize = GlobPtr->datasize;
            SetGlobName((LPCTSTR)GlobPtr->name);
        }
    }
    SetModifiedFlag();
}

long CGlobCtrl::GetDim1Size()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->dim1;
    return -1;
}

void CGlobCtrl::SetDim1Size(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->dim1 = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetDim2Size()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->dim2;
    return -1;
}

void CGlobCtrl::SetDim2Size(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)

```

```

    GlobPtr->dim2 = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetElementSize()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->eltsize;
    return -1;
}

```

```

void CGlobCtrl::SetElementSize(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->eltsize = (short)nNewValue;
    SetModifiedFlag();
}

```

```

long CGlobCtrl::GetType()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->type;
    return -1;
}

```

```

void CGlobCtrl::SetType(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->type = (short)nNewValue;
    SetModifiedFlag();
}

```

```

long CGlobCtrl::GetExtra()
{
    // TODO: Add your property handler here

    if (GlobPtr)
        return GlobPtr->extra;
    return -1;
}

```

```

void CGlobCtrl::SetExtra(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->extra = (short)nNewValue;
    SetModifiedFlag();
}

```

```
long CGlobCtrl::GetDataSize()
```

```
{  
    // TODO: Add your property handler here  
    if (GlobPtr)  
        return GlobPtr->datasize;  
    return 0;  
}
```

```
void CGlobCtrl::SetDataSize(long nNewValue)
```

```
{  
    // TODO: Add your property handler here  
    if (GlobPtr) {  
        GlobPtr->datasize = (short)nNewValue;  
        datasize = GlobPtr->datasize;  
    }  
    SetModifiedFlag();  
}
```

```
BSTR CGlobCtrl::GetGlobName()
```

```
{  
    CString strResult;  
    BYTE nam[17];  
    int i;  
    if (GlobPtr)  
    {  
        for (i=0;i<16;i++) nam[i] = GlobPtr->name[i];  
        nam[16] = '\0';  
        strResult = nam;//m_GlobName;//nam;//GlobPtr->name;  
    }  
    return strResult.AllocSysString();  
}
```

```
void CGlobCtrl::SetGlobName(LPCTSTR lpszNewValue)
```

```
{  
    CString MutexName;  
    // Changing value of GlobName does read of Glob. If found, new properties are seen.  
    RemoveNotify(GlobPtr,GetSafeHwnd());  
    GlobPtr = (!Glob ?) MMFGetGlobPtr(lpszNewValue);  
    if (GlobPtr)  
    {  
        datasize = GlobPtr->datasize;  
        m_GlobName = lpszNewValue;  
    }  
    else  
    {  
        m_GlobName.Empty();  
    }  
}
```

```
SetModifiedFlag();
```

```
long CGlobCtrl::GetFileSize()
```

```
{  
    // TODO: Add your property handler here
```

```
    return m_FileSize;  
}
```

```
void CGlobCtrl::SetFileSize(long nNewValue)
```

```
{  
    // TODO: Add your property handler here  
    SetModifiedFlag();  
}
```

```
BSTR CGlobCtrl::GetFileName()
```

```
{  
    CString strResult;  
    // TODO: Add your property handler here  
    strResult = m_FileName;  
    return strResult.AllocSysString();  
}
```

```
void CGlobCtrl::SetFileName(LPCTSTR lpszNewValue)
```

```
{  
    long retval;  
    CString oldfilename;  
    CString CurrentGlobName;  
  
    CurrentGlobName = GetGlobName();  
    oldfilename = m_FileName;  
    m_FileName.Format("%s", lpszNewValue);  
    m_FileSize = 8192;  
  
    CString x;  
    //AfxMessageBox("Setting file name.");  
    retval = MMFCreate();  
  
    //x.Format("MMFCreate returned: %i", retval);  
    //AfxMessageBox(x);
```

```
    if (retval != OK) {  
        //AfxMessageBox("Set Filename failed.");  
        m_FileName = oldfilename;  
        retval = MMFCreate();  
        if (retval != OK) {  
            m_FileName = "GlobMMF";  
            MMFCreate();  
            SetGlobName(CurrentGlobName);  
            ThrowError(0, "failed to create MMF", 0);  
        }  
        SetGlobName(CurrentGlobName);  
    }
```

```
ThrowError(0,"Invalid FileName.",0);
```

```
}
```

```
//AfxMessageBox("File Name set.");
```

```
SetGlobName(CurrentGlobName);
```

```
SetModifiedFlag();
```

```
long CGlobCtrl::GetStatus()
```

```
{
```

```
// TODO: Add your property handler here
```

```
if (GlobPtr)
```

```
return GlobPtr->status;
```

```
return -1;
```

```
}
```

```
void CGlobCtrl::SetStatus(long nNewValue)
```

```
{
```

```
// TODO: Add your property handler here
```

```
if (GlobPtr) {
```

```
GlobPtr->status = (short)nNewValue;
```

```
SendNotify(GlobPtr,lpView,IDSTATUS,(short)nNewValue);
```

```
}
```

```
SetModifiedFlag();
```

```
}
```

```
long CGlobCtrl::GetCommand()
```

```
{
```

```
if (GlobPtr)
```

```
return GlobPtr->command;
```

```
return -1;
```

```
}
```

```
void CGlobCtrl::SetCommand(long nNewValue)
```

```
{
```

```
// TODO: Add your property handler here
```

```
if (GlobPtr) {
```

```
GlobPtr->command = (short)nNewValue;
```

```
SendNotify(GlobPtr,lpView,IDCOMMAND,(short)nNewValue);
```

```
}
```

```
SetModifiedFlag();
```

```
}
```

```
long CGlobCtrl::GetGlobPtr()
```

```
{
```

```
// TODO: Add your property handler here
```

```
return (long)GlobPtr;
```

```
}
```

```
void CGlobCtrl::SetGlobPtr(long nNewValue)
```

```

{
    // TODO: Add your property handler here
    RemoveNotify(GlobPtr, GetSafeHwnd());
    GlobPtr = (tGlob *)nNewValue; // hope caller know what he's doing.
    datasize = GlobPtr->datasize;
    SetGlobName((LPCSTR)GlobPtr->name);
    SetModifiedFlag();
}

```

```

long CGlobCtrl::GetDataPtr()
{
    // TODO: Add your property handler here

    if(GlobPtr)
        return (long)&(GlobPtr->data); // + sizeof(tGlob);
    else
        return 0;
}

```

```

void CGlobCtrl::SetDataPtr(long nNewValue)
{
    // TODO: Add your property handler here

    // SetModifiedFlag();
}

```

```

long CGlobCtrl::GetAvailSize()
{
    // TODO: Add your property handler here
    if(lpView)
        return m_FileSize - lpView->NextAvail;
    else
        return 0;
}

```

```

void CGlobCtrl::SetAvailSize(long nNewValue)
{
    // TODO: Add your property handler here

    // SetModifiedFlag();
}

```

```

BOOL CGlobCtrl::GetReadOnlyMMF()
{
    // TODO: Add your property handler here
    if(lpView)
        return (BOOL)lpView->ReadOnly;
    else
        return 0;
}

```

```
void CGlobCtrl::SetReadOnlyMMF(BOOL bNewValue)
```

```
{  
    // TODO: Add your property handler here  
    lpView->ReadOnly = (int)bNewValue;  
    SetModifiedFlag();  
}
```

```
BOOL CGlobCtrl::GetNotify()
```

```
{  
    // TODO: Add your property handler here  
    return m_Notify;  
}
```

```
void CGlobCtrl::SetNotify(BOOL bNewValue)
```

```
{  
    // TODO: Add your property handler here  
  
    BOOL ret;  
    HWND hWind;  
    CSingleLock LockMe(GlobLock);  
  
    if(!AmbientUserMode()) {  
        ThrowError(CTL_E_PERMISSIONDENIED, "This property can only be set at runtime.", 0);  
        return;  
    }  
  
    if(!GlobPtr) {  
        ThrowError(CTL_E_PERMISSIONDENIED, "GlobName property is not set.\n Can not register Glob for notification.")  
    }  
  
    return;  
}
```

```
LockMe.Lock(); // waits infinitely for resource to be available.  
               // can use a timeout value as a parameter (ms) if desired.
```

```
hWind = GetSafeHwnd();
```

```
if (bNewValue)  
    ret = AddNotify(GlobPtr, hWind);  
else  
    ret = RemoveNotify(GlobPtr, hWind);
```

```
if (ret)  
    m_Notify = bNewValue;
```

```
LockMe.Unlock();
```

```
SetModifiedFlag();
```

```
}
```



```
long CGlobCtrl::GetValue(long Dim2, long Dim1)
```

```
{
    long l;
    if (GlobPtr)
    {
        l = (Dim2 * GlobPtr->dim1) + Dim1;
        if ((l * GlobPtr->eltsize) < GlobPtr->datasize)
        {
            switch (GlobPtr->eltsize)
            {
                case 4: return GlobPtr->data.Long[l];
                case 2: return GlobPtr->data.Short[l];
                default: return GlobPtr->data.Byte[l];
            }
        }
    }
    return -1;
}
```

```
void CGlobCtrl::SetValue(long Dim2, long Dim1, long nNewValue)
```

```
{
    long l;
    if (! (lpView->ReadOnly))
    {
        if (GlobPtr)
        {
            l = (Dim2 * GlobPtr->dim1) + Dim1;
            if ((l * GlobPtr->eltsize) < GlobPtr->datasize)
            {
                switch (GlobPtr->eltsize)
                {
                    case 4: GlobPtr->data.Long[l] = nNewValue;
                        break;
                    case 2: GlobPtr->data.Short[l] = (short)nNewValue;
                        break;
                    default: GlobPtr->data.Byte[l] = (BYTE) nNewValue;
                        break;
                }
                // notify controls on list of change
                //if (GlobPtr->notify != 0)
                SendNotify(GlobPtr, lpView, IDVALUE, 0);
            }
        }
        //SetModifiedFlag();
    }
}
```

```
*****
MMFCreate: Create file (or just open it) and map a View to it
*****
```

```
long CGlobCtrl::MMFCreate(void)
```

```
{
```

```

struct _stat buf;
int result,i;
CString MMFName;
BOOL FirstMapping;
char buffer[256];
//CString temp;
//BYTE *testView;
//BYTE testRead;
BYTE *MMFLastByte;
long OldFileSize;
long NewFileArea;
long HandleListSize;
DWORD fileretval;
long errcode;

```

```

if (lpView) {
    //reftemp.Format("RefCount-- (MMFCreate) f=%i",f);
    //AfxMessageBox(reftemp);
    lpView->RefCount--;
    UnmapViewOfFile((LPVOID) lpView);
    GlobPtr = NULL;
    lpView=NULL;
    lpLast=NULL;
    CloseHandle(s_hFileMap);
    CloseHandle(f);
    if(GlobLock) delete GlobLock;
    GlobLock = NULL;
    if(MMFLock) delete MMFLock;
    MMFLock = NULL;
}

```

```

//AfxMessageBox("Made it past unmapping stuff");

```

```

fileretval = ::GetFullPathName(m_FileName,254,buffer,NULL);

```

```

if (fileretval == 0) {
    m_FullPath = m_FileName;
} else {
    m_FullPath.Format("%s",(LPCTSTR)buffer);
}

```

```

//AfxMessageBox(m_FullPath);

```

```

MMFName = GetName(m_FullPath);

```

```

// Initialize the Mutex objects
SPX_NOTIFY_MUTEX = MMFName + "NOTIFY";
SPX_MMF_MUTEX = MMFName + "MMF";
GlobLock = new CMutex(false,SPX_NOTIFY_MUTEX,NULL);

```

```
MMFLock = new CMutex(false,SPX_MMF_MUTEX,NULL);
```

```
//AfxMessageBox(MMFName);
```

```
/* Get data associated with file */
```

```
result = _stat( m_FullPath, &buf ); // result will be -1 if file does not exist
```

```
//if the file exists then get its filesize
```

```
if (!result) {
```

```
    if (buf.st_size > m_FileSize)
```

```
        m_FileSize = buf.st_size; // get size of file
```

```
        OldFileSize = buf.st_size;
```

```
    } else {
```

```
        OldFileSize = 0;
```

```
    }
```

```
//DEBUG
```

```
//temp.Format("MMF FileSize = %d\n",m_FileSize);
```

```
//LogErrorString(temp);
```

```
// Create an in-memory memory-mapped file.
```

```
f = CreateFile( m_FullPath,
```

```
    GENERIC_READ | GENERIC_WRITE,
```

```
    FILE_SHARE_READ | FILE_SHARE_WRITE,
```

```
    NULL,
```

```
    OPEN_ALWAYS,
```

```
    FILE_ATTRIBUTE_NORMAL, //FileAttr,
```

```
    NULL);
```

```
if (f==INVALID_HANDLE_VALUE)
```

```
    //AfxMessageBox("Cant open file");
```

```
    //RaiseException(997,0,0,0);
```

```
    //ThrowError(0,"Invalid File Name",0); //ThrowError only works in properties and methods
```

```
    return ERR_INVALIDFILENAME;
```

```
}
```

```
//AfxMessageBox("File opened OK");
```

```
//Grow the file
```

```
SetFilePointer(f,m_FileSize,NULL,FILE_BEGIN);
```

```
SetEndOfFile(f);
```

```
//Error checking?
```

```
s_hFileMap = CreateFileMapping(f, NULL, PAGE_READWRITE, 0, 0, MMFName /*ViewName*/);
```

```
// s_hFileMap = CreateFileMapping((HANDLE) 0xFFFFFFFF, NULL, PAGE_READWRITE, 0, MMFSize, MMFName);
```

```
errcode = GetLastError();
```

```
//reftemp.Format("Error code %i (%i)",errcode,ERROR_ALREADY_EXISTS);
```

```
//AfxMessageBox(reftemp);
```

```
if (errcode == ERROR_ALREADY_EXISTS) {
```

```
    FirstMapping = false;
```

```
} else {
```

```
FirstMapping = true;
```

```
}  
if (s_hFileMap != NULL)
```

```
{  
    // if (GetLastError() == ERROR_ALREADY_EXISTS) MessageBox(myhWnd, _T("MMF Already Exists."), NULL, MB_
```

```
    // File mapping created successfully. Map a view of the file into the address space.
```

```
    lpView = (tControl *)MapViewOfFile(s_hFileMap, FILE_MAP_WRITE | FILE_MAP_READ, 0, 0, 0);
```

```
    if (lpView != NULL)
```

```
    {  
        //fill in 0's if file is expanded
```

```
        if (m_FileSize > OldFileSize) {
```

```
            //LogErrorString("Filling in new memory with 0's\n");
```

```
            MMFLastByte = (BYTE *)((long)lpView + OldFileSize);
```

```
            NewFileArea = m_FileSize - OldFileSize;
```

```
            memset(MMFLastByte, 0, NewFileArea);
```

```
        }
```

```
        //set rest of Glob Attributes
```

```
        if (lpView->nNotifyMaps == 0)
```

```
            lpView->nNotifyMaps = DEF_NOTIFYMAPS;
```

```
            HandleListSize = 32*sizeof(long)*lpView->nNotifyMaps;
```

```
            m_MaxLinks = lpView->nNotifyMaps*32;
```

```
            lpView->Size = m_FileSize; // make global
```

```
            lpView->FirstGlob = sizeof(tControl) + HandleListSize;
```

```
            lpLast = (BYTE *)lpView + m_FileSize - sizeof(tGlob) - sizeof(int);
```

```
            if (!lpView->NextAvail)
```

```
                lpView->NextAvail = sizeof(tControl) + HandleListSize;
```

```
            //reftemp.Format("RefCount++ (MMFCreate) f=%i", f);
```

```
            //AfxMessageBox(reftemp);
```

```
            lpView->RefCount++;
```

```
            // clear out Notify array if this is the first mapping
```

```
            if (FirstMapping) {
```

```
                //AfxMessageBox("First Mapping, Clearing Notify Handles");
```

```
                for (i=0; i<m_MaxLinks; i++)
```

```
                    lpView->NotifyHandle[i] = 0;
```

```
                lpView->RefCount = 1;
```

```
                // Reset globs
```

```
                MMFResetGlobs();
```

```
            }
```

```
            return OK;
```

```
        }
```

```
    else
```

```
    {
```

```
        return ERR_CANT_MAP_VIEW_OF_FILE;
```

```
    }
```

```
}
```

```
else
```

```
{
```

```
return ERR_CANT_CREATE_FILE_MAPPING :-2;
```

```
}  
return ERR_INVALIDFILENAME;
```

```
////////////////////////////////////
```

```
// Create a unique MMF view name from the MMF filename -RK
```

```
////////////////////////////////////
```

```
CString GetName(CString s)
```

```
{  
    int i,j;
```

```
    CString Buffer;
```

```
    int len;
```

```
    //AfxMessageBox(s);
```

```
    j = 0;
```

```
    len = s.GetLength();
```

```
    // load Buffer with spaces
```

```
    for (i=0;i<len;i++)
```

```
        Buffer = Buffer + " ";
```

```
    for (i=0;i<len;i++) {
```

```
        if (s[i] != '\\') {
```

```
            Buffer.SetAt(j,s[i]);
```

```
            j++;
```

```
        }
```

```
    Buffer.TrimRight();
```

```
    // Buffer = s.Right((len-i)-1);
```

```
    Buffer.MakeUpper();
```

```
    return Buffer;
```

```
}
```

```
long CGlobCtrl::MMFOpen(LPCTSTR FileName, long FileAttr, LPCTSTR ViewName, long FileSize)
```

```
{
```

```
    // Create an in-memory memory-mapped file
```

```
    //
```

```
    FileAttr = FILE_ATTRIBUTE_NORMAL; // force it for now
```

```
    f = CreateFile( FileName,
```

```
        GENERIC_READ | GENERIC_WRITE,
```

```
        FILE_SHARE_READ | FILE_SHARE_WRITE,
```

```
        NULL,
```

```
        OPEN_ALWAYS,
```

```
        FileAttr,
```

```
        NULL);
```

```
    if (f==INVALID_HANDLE_VALUE)
```

```
    {
```

```
        // MessageBox(myhWnd,TEXT("Cant open file"),NULL,MB_OK);
```

```
        return 0;
```

```

    }
    s_hFileMap = CreateFileMapping(f, NULL, PAGE_READWRITE, 0, FileSize, ViewName);
    // for memory-only, use "CreateFileMapping((HANDLE) 0xFFFFFFFF, .....
    if (s_hFileMap != NULL)
    {
        // if (GetLastError() == ERROR_ALREADY_EXISTS) MessageBox(myhWnd, __TEXT("MMF Already Exists."), NULL, MB_
        // File mapping created successfully. Map a view of the file into the address space.
        lpView = (tControl *)MapViewOfFile(s_hFileMap, FILE_MAP_READ | FILE_MAP_WRITE, 0, 0, 0);
        if (lpView != NULL)
        {
            // View mapped successfully.
            // To unmap the view: (This protects the data from wayward pointers). "UnmapViewOfFile((LPVOID) lpView);"
            lpView->Size = FileSize; // make global
            lpView->FirstGlob = sizeof( tControl );
            lpLast = (BYTE *)lpView + FileSize - sizeof(tGlob) - sizeof(int);
            if (!lpView->NextAvail) lpView->NextAvail = sizeof( tControl );
            return OK;
        }
    }
    else
    {
        RaiseException(999,0,0,0);
        //MessageBox(myhWnd, __TEXT("Can't map view of file."), NULL, MB_OK);
        return ERR_CANT_MAP_VIEW_OF_FILE;
    }
}
else
{
    RaiseException(998,0,0,0);
    //MessageBox(myhWnd, __TEXT("Can't create file mapping."), NULL, MB_OK);
    return ERR_CANT_CREATE_FILE_MAPPING :-2;
}

return 0;
}

```

```

/*****
MMFRemapView: Close view and re-open as different size
*****/

```

```

long CGlobCtrl::MMFRemapView( long newsize )
{
    return MMFCreat(); // 03/12/98 RBK
}

```

```

/*****
MMFGetGlobIx: Returns index (offset) of Glob if name is found, otherwise 0
*****/

```

```

long CGlobCtrl::MMFGetGlobIx(LPCTSTR GlobName)
{
    tGlob *lpGlob;
    CString name;

```

```

if (GlobName != NULL)
    name = GlobName;
else
    name = "";
if (name.GetLength() == 0)
    return 0;

for (
    lpGlob = (tGlob *)((int)lpView + lpView->FirstGlob);
    lpGlob->size && (int)lpGlob < (int)lpLast;
    lpGlob = (tGlob *)((int)lpGlob + lpGlob->size)
)
    if (!_strnicmp((const char *)GlobName, (const char *)lpGlob->name, Glob_NAME_LENGTH))
    {
        return (int)lpGlob - (int)lpView;
    }
return 0;
}

```

MMFEraseGlob: Clear the Glob to zeroes, but leaving its space still linked in.

```

void CGlobCtrl::MMFEraseGlob( tGlob *B )
{
    int save_size;
    //CString DebugStr;

    if (B)
    {
        save_size = B->size;

        //DebugStr.Format("SaveSize = %i\n", save_size);
        //LogErrorString(DebugStr);

        memset( (BYTE *)B, 0, save_size); // clear all
        B->size = save_size; // restore size for linking past
    }
}

```

MMFFirstGlob: Returns pointer to 1st Glob in the MMF

```

tGlob *CGlobCtrl::MMFFirstGlob()
{
    return (tGlob *)((int)lpView + lpView->FirstGlob);
}

```

MMFNextAvailGlob: Scans from beginning for empty Glob of adequate size. If none, uses NextAvail pointer

```

tGlob *CGlobCtrl::MMFNextAvailGlob( long size )
{
    CString DebugStr;

```

```

int lastchance=(int)lpView + m_FileSize - size - 1;
tGlob *lpGlob;

//DEBUG
//lpGlob = MMFFirstGlob();
//DebugStr.Format("FirstGlobSize: %i\n",lpGlob->size);
//LogErrorString(DebugStr);
//END DEBUG

for ( // chain thru Globs in MMF
lpGlob=MMFFirstGlob(); // first Glob...
lpGlob->size && ((int)lpGlob < lastchance); // if size is nz, within range
lpGlob = (tGlob*)((int)lpGlob + lpGlob->size)
){
//DebugStr.Format("Name: %s size: %i\n", lpGlob->name,lpGlob->size);
//LogErrorString(DebugStr);
if (lpGlob->name[0] == '\0' && lpGlob->size >= size) // if has zeroed-out name and is big enough...
return lpGlob; // return pointer to it
}
// falls thru loop...no empties found
lpGlob = (tGlob*)((int)lpView + lpView->NextAvail); // else get pointer to next one in MMF
if ((int)lpGlob > lastchance) // is there room for it?
lpGlob = 0; // no, return 0
return lpGlob;
}

```

TOOL: MMFGetGlobPtr: Given the GlobName, find and return pointer to Glob, else zero

```

long CGlobCtrl::MMFGetGlobPtr(LPCTSTR GlobName)
{
tGlob *lpGlob;
//CString temp;

for (
lpGlob=MMFFirstGlob();
lpGlob->size && ((int)lpGlob < (int)lpLast);
lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
){
/*
if (LineDebug) {
temp.Format("GlobName: %s\nlpGlob = %i , lpLast = %i , Size = %i\n",((const char *)((tGlob *)lpGlob)
->name),lpGlob,lpLast,lpGlob->size);
LogErrorString(temp);
}
*/
if (!_strnicmp( GlobName, (const char *)((tGlob *)lpGlob)->name, Glob_NAME_LENGTH )) {
//LineDebug = false;
return (int)lpGlob;//true 32-bit pointer
}
}
}
//LineDebug = false;

```



```
return 0;
```

```
}
```

```
void CGlobCtrl::MMFResetGlobs()
```

```
{
```

```
    tGlob *lpGlob;
```

```
    int i;
```

```
    BYTE* temp;
```

```
    long* MapPtr;
```

```
    long datacount;
```

```
    for (
```

```
        lpGlob=MMFFirstGlob();
```

```
        lpGlob->size && (int)lpGlob<(int)lpLast;
```

```
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
```

```
    ){
```

```
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
```

```
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
```

```
        for (i=0;i<lpView->nNotifyMaps;i++){
```

```
            // lpGlob->notifymap[i] = 0;
```

```
            MapPtr = (long*)temp + i;
```

```
            *MapPtr = 0;
```

```
        }
```

```
        lpGlob->command = 0;
```

```
        lpGlob->status = 0;
```

```
    }
```

```
return;
```

```
}
```

```
void CGlobCtrl::MMFClearGlobBits(long index)
```

```
{
```

```
    tGlob *lpGlob;
```

```
    long mapIndex,bitIndex;
```

```
    BYTE* temp;
```

```
    long* MapPtr;
```

```
    long datacount;
```

```
    mapIndex = index/32;
```

```
    bitIndex = index - mapIndex*32;
```

```
    for (
```

```
        lpGlob=MMFFirstGlob();
```

```
        ((int)lpGlob <= (int)lpLast) && lpGlob->size;
```

```
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
```

```
    ){
```

```
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
```

```
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
```

```
        MapPtr = (long*)temp + mapIndex;
```

```
        //lpGlob->notifymap[mapIndex] &= (NOTIFYMASK ^ BitList[bitIndex]);
```

```
        *MapPtr &= (NOTIFYMASK ^ (1<<bitIndex));
```

```
        if(NotifyListIsEmpty(lpGlob)) lpGlob->ptrMap = 0;
```

```

    }
    return;
}

long CGlobCtrl::MMFclearGlobBit(tGlob* GPtr, long index)
{
    tGlob *lpGlob;
    long mapIndex, bitIndex;
    BYTE* temp;
    long* MapPtr;
    long count;
    long datacount;

    count = 0;
    mapIndex = index/32;
    bitIndex = index - mapIndex*32;

    for (
        lpGlob=MMFFirstGlob();
        ((int)lpGlob <= (int)lpLast) && lpGlob->size;
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
    ){
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
        MapPtr = (long*)temp + mapIndex;
        //lpGlob->notifymap[mapIndex] &= (NOTIFYMASK ^ BitList[bitIndex]);
        if(*MapPtr & (1<<bitIndex)) {
            count++;
            if(GPtr == lpGlob) {
                *MapPtr &= (NOTIFYMASK ^ (1<<bitIndex));
                count--;
                if(NotifyListIsEmpty(lpGlob)) lpGlob->ptrMap = 0;
            }
        }
    }
    return count;
}

```

```

BOOL CGlobCtrl::NotifyListIsEmpty(tGlob *lpGlob)
{

```

```

    BYTE* temp;
    long* MapPtr;
    int i;
    BOOL isEmpty;
    long datacount;

```

```

    isEmpty = true;
    datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
    temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
    MapPtr = (long*)temp;

```

```

    for (i=0; i<lpView->nNotifyMaps; i++) {

```

```

    if(*(MapPtr+i)) isEmpty = false;
}

```

```

return isEmpty;
}

```

```

void CGlobCtrl::SetBitMap(tGlob* GlobPtr, long index)
{

```

```

    long mapIndex, bitIndex;

```

```

    long* MapPtr;

```

```

    BYTE* temp;

```

```

    long datacount;

```

```

    mapIndex = index/32;

```

```

    bitIndex = index - mapIndex*32;

```

```

    datacount = GlobPtr->dim1 * GlobPtr->dim2 * GlobPtr->eltsize;

```

```

    if (GlobPtr->ptrMap == 0) {

```

```

        GlobPtr->ptrMap = sizeof(tGlob) + datacount;
    }

```

```

    //GlobPtr->notifymap[mapIndex] |= BitList[bitIndex];

```

```

    temp = (BYTE*)GlobPtr + GlobPtr->ptrMap;

```

```

    MapPtr = (long*)temp + mapIndex;

```

```

    *MapPtr |= (1<<bitIndex); //BitList[bitIndex];
}

```

```

long CGlobCtrl::MMFClose()
{

```

```

    // TODO: Add your dispatch handler code here

```

```

    //reftemp.Format("RefCount-- (MMFClose) f=%i", f);

```

```

    //AfxMessageBox(reftemp);

```

```

    lpView->RefCount--;

```

```

    RemoveNotify(GlobPtr, GetSafeHwnd());

```

```

    UnmapViewOfFile((LPVOID) lpView);

```

```

    lpView=NULL;

```

```

    lpLast=NULL;

```

```

    CloseHandle(s_hFileMap);

```

```

    return CloseHandle(f);
}

```

```

long CGlobCtrl::MMFAddGlob(LPCTSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long T
ype, long Extra)
{

```

```

    int count, ThisGlobSize, LeftOverCount, i;

```

```

    long bitmaploc;

```

```

    BYTE* temp;

```

```

    //CString DebugStr;

```

```

long* MapPtr;

RemoveNotify(GlobPtr,GetSafeHwnd());

if (!Dim2Size)                // if zero, make = 1
    Dim2Size = 1;
if (!Dim1Size)
    Dim1Size = 1;

//DebugStr.Format("Looking for previous Glob: %s\n", GlobName);
//LogErrorString(DebugStr);

GlobPtr = (tGlob *)MMFGetGlobPtr( GlobName ); // first look for one by this name

//DebugStr.Format("Prev Glob Ptr: %i\n", GlobPtr);
//LogErrorString(DebugStr);

if (GlobPtr)                  // found a previous, maybe can reuse space
    MMFEraseGlob( GlobPtr);    // clear it, maybe NextAvail can reuse it
count = Dim2Size * Dim1Size * ElementSize; // calc data size
ThisGlobSize = (sizeof(tGlob) + count + 3) & ~3; // add size of Glob and put on even 4-byte boundary
bmaploc = ThisGlobSize;
ThisGlobSize += sizeof(long) * lpView->nNotifyMaps;

//DebugStr = "Looking for space for Glob\n";
//LogErrorString(DebugStr);

GlobPtr = MMFNextAvailGlob( ThisGlobSize ); // get pointer to area of adequate size

//DebugStr.Format("Found Space at: %i\n", (long)GlobPtr);
//LogErrorString(DebugStr);

while (!GlobPtr)              // no room, make bigger
{
    if(lpView->RefCount > 1) {
        ThrowError(CTL_E_PERMISSIONDENIED, "MMF can not be expanded. Too many connections.\nClose all other appli
cations and try again.");
        return NULL;
    }

    //Debug *****
    //DebugStr.Format("Resizing for %s\n", GlobName);
    //LogErrorString(DebugStr);
    //DebugStr.Format("FileSize Before = %i\n", m_FileSize);
    //LogErrorString(DebugStr);
    // *****

    //LineDebug = true;
    m_FileSize += (ThisGlobSize + 4095); // calc new size of file
    m_FileSize &= ~4095;                // make size a multiple of 4096

    //Debug *****

```

```

//DebugStr.Format("FileSize After = %i\n",m_FileSize);
//LogErrorString(DebugStr);
//*****

MMFCreate(); // unmap/remap view to increase size
GlobPtr = MMFNextAvailGlob( ThisGlobSize ); // get pointer to area of adequate size

//Debug *****
//DebugStr.Format("GlobPtr After Remap: %i\n",GlobPtr);
//LogErrorString(DebugStr);
//*****

SetModifiedFlag(); // properties have changed
}
// setup member variables
datasize = count;
// setup Glob data variables
LeftOverCount = GlobPtr->size - ThisGlobSize; // subtract this size from size that might have been in
GlobPtr->size = ThisGlobSize;
GlobPtr->dim2 = (short)Dim2Size;
GlobPtr->dim1 = (short)Dim1Size;
GlobPtr->eltsize = (short)ElementSize;
GlobPtr->type = (short)Type;
GlobPtr->UOM = UnitsIndex;
GlobPtr->extra = (short)Extra;
GlobPtr->command = 0;
GlobPtr->status = 0;
GlobPtr->ptrMap = 0; //bitmaploc;
temp = (BYTE*)GlobPtr + bitmaploc;
for (i=0;i<lpView->nNotifyMaps;i++) {
    //GlobPtr->notifymap[i] = 0;
    MapPtr = (long*)temp + i;
    *MapPtr = 0;
}
GlobPtr->datasize= count;
memset( GlobPtr->name, 0, Glob_NAME_LENGTH ); // clear name to zeroes
strncpy( (char*)(GlobPtr->name), GlobName, Glob_NAME_LENGTH ); // copy name in
if ((int)GlobPtr == ((int)lpView + lpView->NextAvail)) // if new Glob is at end of file (not reusing ot
her area)
    lpView->NextAvail += ThisGlobSize; // incr nextavail pointer
else // if bytes left over, make new [size] header for empty space left beyond this Glob
{
    if (LeftOverCount > (int)(sizeof(tGlob) + sizeof(long) * lpView->nNotifyMaps)) {
        ((tGlob*)((int)GlobPtr + ThisGlobSize))->size = LeftOverCount; // put a [size] value past this Glob to
reclaim space beyond
    } else {
        GlobPtr->size = ThisGlobSize + LeftOverCount;
    }
}

//DebugStr = "Glob successfully inserted.\n";
//LogErrorString(DebugStr);

```

```

SetModifiedFlag();
return (int)GlobPtr; // returns Glob pointer
}

long CGlobCtrl::MMFAddGlobEx(LPCTSTR GlobName, LPCTSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra)
{
    int status,uom;
    CString Units;
    int count;
    //CString temp;

    count = Dim2Size * Dim1Size * ElementSize;
    if (UnitsName != NULL)
        Units = UnitsName;
    else
        Units = "";
    if (Units.GetLength() != 0) {
        uom = MMFGetGlobIx( Units );
        if (!uom) MMFAddGlob( Units, 0,0,0,0,-1,0); // add unit of measure first
        uom = MMFGetGlobIx( Units );
    } else {
        uom = 0;
    }
    status = MMFAddGlob( GlobName, uom, Dim2Size, Dim1Size, ElementSize, Type, Extra );

    //temp.Format("(%i) AddGlobEx finished successfully for: %s\n",status,GlobName);
    //LogErrorString(temp);

    return status;
}

BOOL CGlobCtrl::GetFirstGlob()
{
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *)((int)lpView + lpView->FirstGlob);
    // HG 980423 SetGlobName((LPCTSTR)GlobPtr->name);
    datasize = GlobPtr->datasize; // HG 980423 copied from SetGlobName
    m_GlobName = GlobPtr->name; // HG 980423 copied from SetGlobName
    SetModifiedFlag(); // cause properties to re-read
    if (GlobPtr->size)
        return true;
    return false;
}

BOOL CGlobCtrl::GetNextGlob()
{
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *)((int)GlobPtr + GlobPtr->size);
    // HG 980423 SetGlobName((LPCTSTR)GlobPtr->name);

```

```

datasize = GlobPtr->datasize; // HG 980423 copied from SetGlobName
m_GlobName = GlobPtr->name; // HG 980423 copied from SetGlobName
SetModifiedFlag(); // cause properties to re-read
if (GlobPtr->size)
    return true;
return false;
}

```

```

void CGlobCtrl::Erase()
{
    CSingleLock LockMe(MMFlock);

    LockMe.Lock();
    RemoveNotify(GlobPtr, GetSafeHwnd());
    MMFEraseGlob( GlobPtr );
    LockMe.Unlock();
}

```

```

void CGlobCtrl::MMFErase()
{
    CSingleLock LockMe(MMFlock);

    LockMe.Lock();
    if (lpView)
    {
        int size;
        int nmaps;

        RemoveNotify(GlobPtr, GetSafeHwnd());
        size = lpView->Size;
        nmaps = lpView->nNotifyMaps;
        memset( lpView, 0, size );
        lpView->nNotifyMaps = nmaps;
        lpView->NextAvail = lpView->FirstGlob = sizeof( tControl ) + 32*sizeof(long)*nmaps;
        lpView->Size = size;
    }
    LockMe.Unlock();
}

```

```

long CGlobCtrl::GetNotifyList(long index)
{
    BYTE* temp;
    long* MapPtr;

    if (lpView) {
        if ((index < 0) || (index >= lpView->nNotifyMaps)) return 0;
        // TODO: Add your property handler here
        if (GlobPtr && GlobPtr->ptrMap) {
            temp = (BYTE*)GlobPtr + GlobPtr->ptrMap;
            MapPtr = (long*)temp + index;

```

```
return *MapPtr;
```

```
}
```

```
return 0;
```

```
void CGlobCtrl::SetNotifyList(long index, long nNewValue)
```

```
{
```

```
// TODO: Add your property handler here
```

```
SetModifiedFlag();
```

```
}
```

```
BOOL CGlobCtrl::AddNotify(tGlob* GlobPtr,HWND my_hWnd)
```

```
{
```

```
int index;
```

```
// add my hWnd to notify list
```

```
if(GlobPtr){
```

```
index = FindHandle(my_hWnd); // look for a previous entry
```

```
if (index != -1){ // if we find one, dont add another!
```

```
SetBitMap(GlobPtr,index); // RK 042498
```

```
return true;
```

```
}
```

```
// didnt find one so make one
```

```
index = FindHandle(0); // look for first 0 entry
```

```
if (index != -1){ // make sure there is one available
```

```
lpView->NotifyHandle[index] = my_hWnd;
```

```
m_Notify = true;
```

```
/*
```

```
CString Temp;
```

```
Temp.Format("index = %d Power(index) = %d",index,Power(index));
```

```
AfxMessageBox(Temp);
```

```
*/
```

```
SetBitMap(GlobPtr,index); // add ref to notify list
```

```
return true;
```

```
}
```

```
}
```

```
ThrowError(CTL_E_OUTOFMEMORY,"Out of memory in MMF. Can not register Glob for notification.");
```

```
return false;
```

```
}
```

```
BOOL CGlobCtrl::RemoveNotify(tGlob* GlobPtr, HWND my_hWnd)
```

```
{
```

```
int index;
```

```
// remove my hWnd from the notify list
```

```
if(GlobPtr){
```

```
m_Notify = false; // HG 980423 clear notify flag in any case
```

```
index = FindHandle(my_hWnd); // look for handle in list
```



```

if (index == -1)           // not there!
    return true;          // dont need to remove anything

```

```

lpView->NotifyHandle[index] = 0; // remove entry from list
MMFClearGlobBits(index);
//GlobPtr->notify &= (NOTIFYMASK ^ BitList[index]); // remove ref from notify map
return true;

```

```

}
return false;

```

```

BOOL CGlobCtrl::RemoveNotifyX(tGlob* GPtr, HWND my_hWnd)

```

```

{
    int index;
    long count;

    // remove my hwnd from the notify list
    if(GlobPtr){
        index = FindHandle(my_hWnd); // look for handle in list
        if (index == -1)           // not there!
            return true;          // dont need to remove anything
        count = MMFClearGlobBit(GPtr,index);
        if (count == 0){
            lpView->NotifyHandle[index] = 0;
            m_Notify = false;
        }
        return true;
    }
    return false;
}

```

```

int CGlobCtrl::FindHandle(HWND my_hWnd)

```

```

{
    int i;

    for (i=0; i<m_MaxLinks; i++)
        if (lpView->NotifyHandle[i] == my_hWnd) return i;

    return -1;
}

```

```

long CGlobCtrl::GetNotifyHandle(short index)

```

```

{
    // TODO: Add your property handler here
    if ((lpView) && (index < m_MaxLinks) && (index >= 0))
        return (long)lpView->NotifyHandle[index];

    return -1;
}

```

```

void CGlobCtrl::SetNotifyHandle(short index, long nNewValue)

```

```

{
    // TODO: Add your property handler here

    SetModifiedFlag();

    long Power(int operand)
    {
        int i;
        long value;

        if (operand == 0) {
            value = 1;
        } else {
            value = 1;
            for(i=1; i<=operand; i++)
                value *=2;
        }

        return value;
    }

    void CGlobCtrl::SendNotify(tGlob* GlobPtr, tControl* lpView, short IDProp, short PropValue)
    {
        // TODO: Add your dispatch handler code here
        int i, mapIndex, bitIndex;
        int results;
        HWND mHwnd;
        tGlobMsg msg;
        long* tempmsg;
        long GlobID;
        int count;
        long map;

        count = 0;

        // set up message to send for notify
        msg.PropID = IDProp;
        msg.Value = PropValue;
        tempmsg = (long*)&msg;
        GlobID = (long)GlobPtr - (long)lpView;

        mHwnd = GetSafeHwnd();

        if(m_AutoNotify) {
            if(GlobPtr && GlobPtr->ptrMap) {
                for (mapIndex=0; mapIndex<lpView->nNotifyMaps; mapIndex++) {
                    map = GetNotifyList(mapIndex);
                    if (map != 0) count++;
                    for (bitIndex = 0; bitIndex<32; bitIndex++) {
                        i = mapIndex*32+bitIndex;
                        if((map & (1<<bitIndex)) && (mHwnd != lpView->NotifyHandle[i])) {
                            results = ::PostMessage(lpView->NotifyHandle[i], USER_VALUECHANGED, *tempmsg, GlobID);
                            if (!results) // if the handle is invalid then remove it from the list

```

```
RemoveNotify(GlobPtr,lpView->NotifyHandle[i]);
```

```
if (count == 0) GlobPtr->ptrMap = 0;
```

```
void CGlobCtrl::OnFinalRelease()
```

```
// TODO: Add your specialized code here and/or call the base class
```

```
RemoveNotify(GlobPtr,GetSafeHwnd());
```

```
COleControl::OnFinalRelease();
```

```
long CGlobCtrl::OnValueChanged(UINT lParam, LONG rParam)
```

```
//unpack lParam for PropID and Value
```

```
tGlobMsg* msg;
```

```
msg = (tGlobMsg*)&lParam;
```

```
FileChange(msg->PropID,msg->Value,rParam);
```

```
return 0;
```

```
short CGlobCtrl::GetByteValue()
```

```
// TODO: Add your property handler here
```

```
// returns a byte (short was the only option in the wizard :)
```

```
if (GlobPtr)
```

```
return GlobPtr->data.Byte[0];
```

```
return 0;
```

```
void CGlobCtrl::SetByteValue(short nNewValue)
```

```
// TODO: Add your property handler here
```

```
if ((GlobPtr) && !(lpView->ReadOnly)) {
```

```
GlobPtr->data.Byte[0] = (BYTE)nNewValue;
```

```
// notify controls on list of change
```

```
//if(GlobPtr->notify !=0)
```

```
SendNotify(GlobPtr,lpView,IDVALUE,0);
```

```
SetModifiedFlag();
```

```
short CGlobCtrl::GetAbValue(long index)
```

```
// TODO: Add your property handler here
```

```

    if ((GlobPtr) && (index < GlobPtr->datasize))
        return GlobPtr->data.Byte[index];
    return 0;
}

void CGlobCtrl::SetAbValue(long index, short nNewValue)
{
    // TODO: Add your property handler here
    if (!(lpView->ReadOnly))
        if ((GlobPtr) && (index < GlobPtr->datasize)) {
            GlobPtr->data.Byte[index] = (BYTE)nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
        }
    //SetModifiedFlag(); -RK not needed for non persistent properties
}

```

```

long CGlobCtrl::GetLValue()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->data.Long[0];
    return 0;
}

```

```

void CGlobCtrl::SetLValue(long nNewValue)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && !(lpView->ReadOnly)) {
        if (GlobPtr->eltsize == 4)
            GlobPtr->data.Long[0] = nNewValue;

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
        SendNotify(GlobPtr,lpView,IDVALUE,0);
    }
    //SetModifiedFlag();
}

```

```

long CGlobCtrl::GetAlValue(long index)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && ((index * sizeof(long)) < GlobPtr->datasize))
        return GlobPtr->data.Long[index];
    return 0;
}

```

```

void CGlobCtrl::SetAlValue(long index, long nNewValue)
{
    // TODO: Add your property handler here
}

```

```

if (!(lpView->ReadOnly))
    if ((GlobPtr) && ((index * sizeof(long)) < GlobPtr->datasize)) {
        GlobPtr->data.Long[index] = nNewValue;

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
    }
    //SetModifiedFlag();

```

```

short CGlobCtrl::GetIValue()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->data.Short[0];
    return 0;
}

```

```

void CGlobCtrl::SetIValue(short nNewValue)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && !(lpView->ReadOnly)) {
        if (GlobPtr->elsize >= 2)
            GlobPtr->data.Short[0] = nNewValue;

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
    }
    //SetModifiedFlag();
}

```

```

short CGlobCtrl::GetAiValue(long index)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && ((index * sizeof(short)) < GlobPtr->datasize))
        return GlobPtr->data.Short[index];
    return 0;
}

```

```

void CGlobCtrl::SetAiValue(long index, short nNewValue)
{
    // TODO: Add your property handler here
    if (!(lpView->ReadOnly))
        if ((GlobPtr) && ((index * sizeof(short)) < GlobPtr->datasize)) {
            GlobPtr->data.Short[index] = nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
                SendNotify(GlobPtr,lpView,IDVALUE,0);
        }
}

```

```
//SetModifiedFlag();
```

```
long CGlobCtrl::GetValueSD(long n)
```

```
// TODO: Add your property handler here
```

```
if (GlobPtr)
```

```
{
```

```
if ((n * GlobPtr->eltsize) < GlobPtr->datasize)
```

```
switch (GlobPtr->eltsize)
```

```
{
```

```
case 4: return GlobPtr->data.Long[n];
```

```
case 2: return GlobPtr->data.Short[n];
```

```
default: return GlobPtr->data.Byte[n];
```

```
}
```

```
}
```

```
}
```

```
return -1;
```

```
void CGlobCtrl::SetValueSD(long n, long nNewValue)
```

```
{
```

```
// TODO: Add your property handler here
```

```
if (!(lpView->ReadOnly))
```

```
{
```

```
if (GlobPtr)
```

```
{
```

```
if ((n * GlobPtr->eltsize) < GlobPtr->datasize)
```

```
{
```

```
switch (GlobPtr->eltsize)
```

```
{
```

```
case 4: GlobPtr->data.Long[n] = nNewValue;
```

```
break;
```

```
case 2: GlobPtr->data.Short[n] = (short)nNewValue;
```

```
break;
```

```
default: GlobPtr->data.Byte[n] = (BYTE) nNewValue;
```

```
break;
```

```
}
```

```
// notify controls on list of change
```

```
//if(GlobPtr->notify !=0)
```

```
SendNotify(GlobPtr,lpView,IDVALUE,0);
```

```
}
```

```
}
```

```
}
```

```
//SetModifiedFlag();
```

```
}
```

```
BSTR CGlobCtrl::GetStrValue()
```

```
{
```

```
CString strResult;
```

```
if (GlobPtr)
```

```

    strResult = (GlobPtr->data.Byte);
    // TODO: Add your property handler here

    return strResult.AllocSysString();
}

void CGlobCtrl::SetStrValue(LPCTSTR lpszNewValue)
{
    // TODO: Add your property handler here
    //CString strResult(lpszNewValue);
    int i;
    int size;

    if (!(!lpView->ReadOnly) && (GlobPtr)) {
        size = strlen(lpszNewValue);
        for(i=0; (i < size) && (i < (GlobPtr->datasize-1)); i++) {
            GlobPtr->data.Byte[i] = lpszNewValue[i];
        }
        GlobPtr->data.Byte[i] = "\0";

        // notify controls on list of change
        //if(GlobPtr->notify != 0)
        SendNotify(GlobPtr, lpView, IDVALUE, 0);
    }

    //SetModifiedFlag();
}

void CGlobCtrl::Insert(long value, long index)
{
    // TODO: Add your dispatch handler code here
    LPBYTE source;
    LPBYTE dest;
    LONG size;
    long datacount;

    // exit if index is beyond range or Glob isnt setup
    if (!GlobPtr) return;
    datacount = GlobPtr->dim1 * GlobPtr->dim2 * GlobPtr->eltsize;
    if (((index * GlobPtr->eltsize) >= datacount) || (index < 0))
        return;

    if ((GlobPtr) && !(!lpView->ReadOnly)) {
        source = GlobPtr->data.Byte + index * GlobPtr->eltsize;
        dest = source + GlobPtr->eltsize;
        size = datacount - (index + 1) * GlobPtr->eltsize;

        //move data up (memmove handles overlapping memory regions)
        memmove(dest, source, size);

        //insert new data element
    }
}

```

```

switch (GlobPtr->eltsize)
{
    case 4: GlobPtr->data.Long[index] = value;
        break;
    case 2: GlobPtr->data.Short[index] = (short)value;
        break;
    default: GlobPtr->data.Byte[index] = (BYTE) value;
        break;
}
// notify controls on list of change
//if(GlobPtr->notify != 0)
    SendNotify(GlobPtr,lpView,IDVALUE,0);
}
}

```

BSTR CGlobCtrl::GetFullPath()

```

{
    CString strResult;
    // TODO: Add your property handler here
    strResult = m_FullPath;
    return strResult.AllocSysString();
}

```

void CGlobCtrl::SetFullPath(LPCTSTR lpszNewValue)

```

{
    // TODO: Add your property handler here
    SetModifiedFlag();
}

```

long CGlobCtrl::ResizeMMF(long NewSize)

```

{
    // TODO: Add your dispatch handler code here
    return MMFRemapView(NewSize);
}

```

long CGlobCtrl::SyncFileSize()

```

{
    if (m_FileSize != lpView->Size) {
        // mapviews are not synchronized so remap
        MMFRemapView(lpView->Size);
    }
    return 0;
}

```

void LogErrorString(CString errstr)

```

{
    if (!LOG_ERRORS) return;

    FILE *f;

    f = fopen("C:\\GlobErr.Log","a");

```



```

    fwrite(errstr,1,errstr.GetLength(),f);
    fclose(f);
}

```

```

void CGlobCtrl::SendNotifyX(short NotifyID = 0, short Value = 0)
{

```

```

    // TODO: Add your dispatch handler code here

```

```

    int i, mapIndex, bitIndex;

```

```

    int results;

```

```

    HWND mHwnd;

```

```

    tGlobMsg msg;

```

```

    long* tempmsg;

```

```

    long GlobID;

```

```

    // set up message to send for notify

```

```

    msg.PropID = NotifyID;

```

```

    msg.Value = Value;

```

```

    tempmsg = (long*)&msg;

```

```

    GlobID = (long)GlobPtr - (long)lpView;

```

```

    mHwnd = GetSafeHwnd();

```

```

    if(GlobPtr) {

```

```

        for (mapIndex=0; mapIndex<lpView->nNotifyMaps; mapIndex++) {

```

```

            for (bitIndex = 0; bitIndex<32; bitIndex++) {

```

```

                i = mapIndex*32+bitIndex;

```

```

                if((GetNotifyList(mapIndex) & (1<<bitIndex)) && (mHwnd != lpView->NotifyHandle[i])) {

```

```

                    results = ::PostMessage(lpView->NotifyHandle[i], USER_VALUECHANGED, *tempmsg, GlobID);

```

```

                    if (!results) // if the handle is invalid then remove it from the list

```

```

                        RemoveNotify(GlobPtr, lpView->NotifyHandle[i]);

```

```

                }

```

```

            }

```

```

        }

```

```

    }

```

```

BOOL CGlobCtrl::GetAutoSendNotify()
{

```

```

    // TODO: Add your property handler here

```

```

    return m_AutoNotify;
}

```

```

void CGlobCtrl::SetAutoSendNotify(BOOL bNewValue)
{

```

```

    // TODO: Add your property handler here

```

```

    m_AutoNotify = bNewValue;

```

```

    SetModifiedFlag();
}

```

```
BOOL CGlobCtrl::OnSetExtent(LPSIZEL lpSizeL)
```

```
{  
    // TODO: Add your specialized code here and/or call the base class
```

```
    return false; //COleControl::OnSetExtent(lpSizeL);  
}
```

```
long CGlobCtrl::GetNHandles()
```

```
{  
    // TODO: Add your property handler here
```

```
    return m_MaxLinks;  
}
```

```
void CGlobCtrl::SetNHandles(long nNewValue)
```

```
{  
    // TODO: Add your property handler here
```

```
    SetModifiedFlag();  
}
```

```
long CGlobCtrl::GetNNotifyMaps()
```

```
{  
    // TODO: Add your property handler here  
    if (lpView)  
        return lpView->nNotifyMaps;
```

```
    return 0;  
}
```

```
void CGlobCtrl::SetNNotifyMaps(long nNewValue)
```

```
{  
    // TODO: Add your property handler here
```

```
    SetModifiedFlag();  
}
```

```
BOOL CGlobCtrl::FormatMMF(long NotifyLimit)
```

```
{  
    // TODO: Add your dispatch handler code here  
    int x;
```

```
    if (lpView) {
```

```
        if (lpView->RefCount > 1) {
```

```
            ThrowError(CTL_E_PERMISSIONDENIED, "Sharing violation. Can not reformat MMF.");
```

```
            return false;
```

```
        }
```

```

    if (NotifyLimit < 32) NotifyLimit = 32;
    x = (NotifyLimit-1)/32 + 1;
    lpView->nNotifyMaps = x;
    m_MaxLinks = x*32;

    m_FileSize += (m_MaxLinks*sizeof(long)+ 4095); // calc new size of file
    m_FileSize &= ~4095; // make size a multiple of 4096

```

```

    MMFErase();
    if(MMFCreate() == OK) return TRUE;
}
return FALSE;
}

```

```

long CGlobCtrl::GetGlobSize()
{
    // TODO: Add your property handler here

    if(GlobPtr) return GlobPtr->size;
    return sizeof(tGlob);
}

```

```

void CGlobCtrl::SetGlobSize(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

```

```

long CGlobCtrl::GetRefCount()
{
    // TODO: Add your property handler here
    if(lpView) return lpView->RefCount;

    return 0;
}

```

```

void CGlobCtrl::SetRefCount(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

```

```

BSTR CGlobCtrl::GetVersion()
{
    CString strResult;
    // TODO: Add your property handler here
    strResult = VERSION;
    return strResult.AllocSysString();
}

```

```
void CGlobCtrl::SetVersion(LPCTSTR lpszNewValue)
```

```
{  
    // TODO: Add your property handler here
```

```
    SetModifiedFlag();  
}
```

```
BSTR CGlobCtrl::GetUOM()
```

```
{  
    CString strResult;  
    // TODO: Add your property handler here  
    tGlob* temp;
```

```
    if ((GlobPtr) && (GlobPtr->UOM) && lpView){  
        temp = (tGlob*)((int)lpView + GlobPtr->UOM);  
        strResult = temp->name;
```

```
    } else {  
        strResult = "";
```

```
    }  
    return strResult.AllocSysString();  
}
```

```
void CGlobCtrl::SetUOM(LPCTSTR lpszNewValue)
```

```
{  
    // TODO: Add your property handler here  
    int uom;
```

```
    uom = MMFGetGlobIx(lpszNewValue);
```

```
    if (GlobPtr) GlobPtr->UOM = uom;  
    SetModifiedFlag();  
}
```

```
BSTR CGlobCtrl::GetLink()
```

```
{  
    CString strResult;
```

```
    // TODO: Add your property handler here  
    tGlob* temp;
```

```
    if ((GlobPtr) && (GlobPtr->link) && lpView){  
        temp = (tGlob*)((int)lpView + GlobPtr->link);  
        strResult = temp->name;
```

```
    } else {  
        strResult = "";
```

```
    }  
    return strResult.AllocSysString();  
}
```

```
void CGlobCtrl::SetLink(LPCTSTR lpszNewValue)
```

```

{
    // TODO: Add your property handler here
    int link;
    CString newval;

    if (lpzNewValue != NULL)
        newval = lpzNewValue;
    else
        newval = "";

    link = MMFGetGlobIx(newval);
    if (GlobPtr) GlobPtr->link = link;
    SetModifiedFlag();
}

long CGlobCtrl::IndexOf(LPCTSTR GlobName)
{
    // TODO: Add your dispatch handler code here
    long index;

    index = MMFGetGlobIx(GlobName);
    if (index) index = -1;
    return index;
}

BOOL CGlobCtrl::GetNotifyOnChange(LPCTSTR GlobName)
{
    // TODO: Add your property handler here
    tGlob* GPtr;
    HWND hWind;
    long index;
    long mapIndex;
    long bitIndex;
    BOOL ret;
    BYTE* temp;
    long* MapPtr;

    hWind = GetSafeHwnd();
    ret = false;

    if (!pView) return false;

    if (GlobName[0] == '\\0') {
        GPtr = GlobPtr;
    } else {
        GPtr = (tGlob*)MMFGetGlobPtr(GlobName);
    }

    if (GPtr && GPtr->ptrMap) {
        index = FindHandle(hWind);
        if (index != -1) {
            mapIndex = index/32;

```

```

        bitIndex = index - mapIndex*32;
        temp = (BYTE*)GPtr + GPtr->ptrMap;
        MapPtr = (long*)temp + mapIndex;
        if (*MapPtr & (1<<bitIndex)) ret = true;
    }
}

return ret;
}

void CGlobCtrl::SetNotifyOnChange(LPCTSTR GlobName, BOOL bNewValue)
{
    // TODO: Add your property handler here

    BOOL ret;
    HWND hWind;
    tGlob* GPtr;

    if (GlobName[0] == '\0') {
        GPtr = GlobPtr;
    } else {
        GPtr = (tGlob*)MMFGetGlobPtr(GlobName);

        if (GPtr) {
            CSingleLock LockMe(GlobLock);

            if (!AmbientUserMode()) {
                ThrowError(CTL_E_PERMISSIONDENIED, "This property can only be set at runtime.", 0);
                return;
            }

            LockMe.Lock(); // waits infinitely for resource to be available.
                           // can use a timeout value as a parameter (ms) if desired.

            hWind = GetSafeHwnd();

            if (bNewValue) {
                ret = AddNotify(GPtr, hWind);
            } else {
                ret = RemoveNotifyX(GPtr, hWind);
            }

            LockMe.Unlock();
        }

        SetModifiedFlag();
    }

    long CGlobCtrl::SetVisible()
    {

```

```

HRESULT hresult;
IDispatch FAR* pdisp = (IDispatch FAR*)NULL;
DISPID dispid;
OLECHAR FAR* szVisible = L"Visible";
OLECHAR FAR* szTabStop = L"TabStop";
DISPPARAMS disparams;
DISPID MyDispid = DISPID_PROPERTYPUT;
VARIANTARG myarg[1];

```

```

disparams.rgvarg = myarg;
disparams.rgvarg[0].vt = VT_BOOL;
disparams.rgvarg[0].boolVal = FALSE; //MFC help says this fieldname is actually "bool"... yea right!
disparams.rgdispidNamedArgs = &MyDispid;
disparams.cArgs = 1;
disparams.cNamedArgs = 1;

```

```

pdisp = GetExtendedControl();
hresult = DISP_E_UNKNOWNINTERFACE;
if (pdisp) {
    //set visible to false
    hresult = pdisp->GetIDsOfNames(IID_NULL,&szVisible,1,LOCALE_USER_DEFAULT,&dispid);
    if (hresult == S_OK) {
        hresult = pdisp->Invoke(dispid,IID_NULL,LOCALE_USER_DEFAULT,DISPATCH_PROPERTYPUT,
            &disparams,NULL,NULL,NULL);
    }
    //set TabStop to false
    hresult = pdisp->GetIDsOfNames(IID_NULL,&szTabStop,1,LOCALE_USER_DEFAULT,&dispid);
    if (hresult == S_OK) {
        hresult = pdisp->Invoke(dispid,IID_NULL,LOCALE_USER_DEFAULT,DISPATCH_PROPERTYPUT,
            &disparams,NULL,NULL,NULL);
    }
    pdisp->Release();
}
return (long)hresult;
}

```

```
// Copyright 1998, 1999 SPX Corporation
#if !defined(AFX_GLOB_H__5F20D2DC_788C_11D1_9A9B_020701045A6B__INCLUDED_)
#define AFX_GLOB_H__5F20D2DC_788C_11D1_9A9B_020701045A6B__INCLUDED_
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
// Glob.h : main header file for GLOB.DLL
```

```
#if !defined(_AFXCTL_H_)
#error include 'afxctl.h' before including this file
#endif
```

```
#include "resource.h" // main symbols
```

```
////////////////////////////////////
// CGlobApp : See Glob.cpp for implementation.
```

```
#define Glob_NAME_LENGTH 16
#define DEF_NOTIFYMAPS 8
#define NOTIFYMASK -1
#define OK 0
#define ERR_CANT_CREATE_FILE_MAPPING -2
#define ERR_CANT_MAP_VIEW_OF_FILE -1
#define ERR_INVALID_Glob_REFERENCE -3
#define ERR_INVALIDFILENAME 999
#define MEM_ALLOC 4096
#define IMAX_STRING 256
#define MMF_INTERCOM_MMF __TEXT("MMF_INTERCOM")
```

```
// FLAG VALUES TO USE IN MMFGETGlobPARAM AND MMFSETGlobPARAM
// USE ACTUAL BYTE OFFSETS FOR FASTER ACCESS
```

```
#define Glob_DIM2 4
#define Glob_DIM1 6
#define GlobELTSIZE 8
#define Glob_TYPE 10
#define Glob_PARAM 12 // addl data
#define Glob_DATASIZE 14 // addl
```

```
typedef struct
```

```
{
    long Size;
    int FirstGlob;
    int NextAvail;
    int ReadOnly; // is MMF ReadOnly right now?
    int RefCount;
    int nNotifyMaps;
    int Data[ 10 ]; // spare
    HWND NotifyHandle[0]; // hwnd for windows to notify of changes
} tControl;
```



```

typedef struct
{
    int size;
    BYTE name[ Glob_NAME_LENGTH ];
    short dim2;    // 2nd dimension
    short dim1;    // 1st dimension
    short eltsize; // byte size of each array element
    short type;    // type of array element
    short extra;   // addl data. Waveforms use for Actual Length, etc.
    short command; // command to the device
    short status;  // status from the device
    short datasize; // addl
    long UOM;      // unit of measure link, if any
    long link;     // offset of parameter Glob, if any
    //long notifymap[DEF_NOTIFYMAPS]; // bitmap used to indicate who to notify if changed
    long ptrMap;
    union
    {
        long Long[0];
        short Short[0];
        BYTE Byte[0];
    } data;
} tGlob;

typedef struct
{
    short PropID;
    short Value;
} tGlobMsg;

class CGlobApp : public COleControlModule
{
public:
    BOOL InitInstance();
    int ExitInstance();
};

extern const GUID CDECL _tlid;
extern const WORD _wVerMajor;
extern const WORD _wVerMinor;

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_GLOB_H__5F20D2DC_78BC_11D1_9A9B_020701045A6B__INCLUDED)

```

```
// Copyright 1998, 1999 SPX Corporation
// GlobPpg.cpp : Implementation of the CGlobPropPage property page class.
```

```
#include "stdafx.h"
#include "Glob.h"
#include "GlobPpg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__
#endif
```

```
IMPLEMENT_DYNCREATE(CGlobPropPage, COlePropertyPage)
```

```
////////////////////////////////////
// Message map
```

```
BEGIN_MESSAGE_MAP(CGlobPropPage, COlePropertyPage)
    //{{AFX_MSG_MAP(CGlobPropPage)
    // NOTE - ClassWizard will add and remove message map entries
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// Initialize class factory and guid
```

```
IMPLEMENT_OLECREATE_EX(CGlobPropPage, "GLOB.GlobPropPage.1",
    0x5f20d2d7, 0x788c, 0x11d1, 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b)
```

```
////////////////////////////////////
// CGlobPropPage::CGlobPropPageFactory::UpdateRegistry -
// Adds or removes system registry entries for CGlobPropPage
```

```
BOOL CGlobPropPage::CGlobPropPageFactory::UpdateRegistry(BOOL bRegister)
{
    if (bRegister)
        return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
            m_clsid, IDS_GLOB_PPG);
    else
        return AfxOleUnregisterClass(m_clsid, NULL);
}
```

```
////////////////////////////////////
// CGlobPropPage::CGlobPropPage - Constructor
```

```

CGlobPropPage::CGlobPropPage() :
    COlePropertyPage(IDD, IDS_GLOB_PPG_CAPTION)
{
   //{{AFX_DATA_INIT(CGlobPropPage)
    // NOTE: ClassWizard will add member initialization here
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_DATA_INIT
}

////////////////////////////////////
// CGlobPropPage::DoDataExchange - Moves data between page and properties

void CGlobPropPage::DoDataExchange(CDataExchange* pDX)
{
   //{{AFX_DATA_MAP(CGlobPropPage)
    // NOTE: ClassWizard will add DDP, DDX, and DDV calls here
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}

////////////////////////////////////
// CGlobPropPage message handlers

```

```

// Copyright 1998, 1999 SPX Corporation
#ifdef AFX_GLOBCTL_H__5F20D2E4_788C_11D1_9A9B_020701045A6B__INCLUDED_
#define AFX_GLOBCTL_H__5F20D2E4_788C_11D1_9A9B_020701045A6B__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

```

```

// GlobCtl.h : Declaration of the CGlobCtrl ActiveX Control class.

```

```

////////////////////////////////////
// CGlobCtrl : See GlobCtl.cpp for implementation.

```

```

#include <afxmt.h>
#include <memory.h>
#include <string.h>

```

```

#define IDVALUE 1
#define IDSTATUS 2
#define IDCMMAND 3
#define VERSION "1.2i"

```

```

class CGlobCtrl : public COleControl
{
    DECLARE_DYNCREATE(CGlobCtrl)

```

```

// Constructor

```

```

public:
    CGlobCtrl();

```

```

// Overrides

```

```

// ClassWizard generated virtual function overrides

```

```

//{{AFX_VIRTUAL(CGlobCtrl)

```

```

public:

```

```

virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);

```

```

virtual void DoPropExchange(CPropExchange* pPX);

```

```

virtual void OnResetState();

```

```

virtual void OnFinalRelease();

```

```

virtual BOOL OnSetExtent(LPSIZEL lpSizeL);

```

```

//}}AFX_VIRTUAL

```

```

// Implementation

```

```

protected:

```

```

    ~CGlobCtrl();

```

```

BEGIN_OLEFACTORY(CGlobCtrl)

```

```

    virtual BOOL VerifyUserLicense();

```

```

    virtual BOOL GetLicenseKey(DWORD, BSTR FAR*);

```

```

END_OLEFACTORY(CGlobCtrl)

```

```

DECLARE_OLETYPELIB(CGlobCtrl) // GetTypeInfo

```

```

DECLARE_PROPPAGEIDS(CGlobCtrl) // Property page IDs

```

```

DECLARE_OLECTLTYPE(CGlobCtrl) // Type name and misc status

```

```

// Message maps
//{{AFX_MSG(CGlobCtrl)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

// Dispatch maps
//{{AFX_DISPATCH(CGlobCtrl)
afx_msg long GetGlobIndex();
afx_msg void SetGlobIndex(long nNewValue);
afx_msg long GetDim1Size();
afx_msg void SetDim1Size(long nNewValue);
afx_msg long GetDim2Size();
afx_msg void SetDim2Size(long nNewValue);
afx_msg long GetElementSize();
afx_msg void SetElementSize(long nNewValue);
afx_msg long GetType();
afx_msg void SetType(long nNewValue);
afx_msg long GetExtra();
afx_msg void SetExtra(long nNewValue);
afx_msg long GetDataSize();
afx_msg void SetDataSize(long nNewValue);
afx_msg BSTR GetGlobName();
afx_msg void SetGlobName(LPCTSTR lpszNewValue);
afx_msg long GetFileSize();
afx_msg void SetFileSize(long nNewValue);
afx_msg BSTR GetFileName();
afx_msg void SetFileName(LPCTSTR lpszNewValue);
afx_msg long GetStatus();
afx_msg void SetStatus(long nNewValue);
afx_msg long GetCommand();
afx_msg void SetCommand(long nNewValue);
afx_msg long GetGlobPtr();
afx_msg void SetGlobPtr(long nNewValue);
afx_msg long GetDataPtr();
afx_msg void SetDataPtr(long nNewValue);
afx_msg long GetAvailSize();
afx_msg void SetAvailSize(long nNewValue);
afx_msg BOOL GetReadOnlyMMF();
afx_msg void SetReadOnlyMMF(BOOL bNewValue);
afx_msg BOOL GetNotify();
afx_msg void SetNotify(BOOL bNewValue);
afx_msg short GetByteValue();
afx_msg void SetByteValue(short nNewValue);
afx_msg long GetLValue();
afx_msg void SetLValue(long nNewValue);
afx_msg short GetIValue();
afx_msg void SetIValue(short nNewValue);
afx_msg BSTR GetStrValue();
afx_msg void SetStrValue(LPCTSTR lpszNewValue);
afx_msg BSTR GetFullPath();
afx_msg void SetFullPath(LPCTSTR lpszNewValue);

```

```

afx_msg BOOL GetAutoSendNotify();
afx_msg void SetAutoSendNotify(BOOL bNewValue);
afx_msg long GetNHandles();
afx_msg void SetNHandles(long nNewValue);
afx_msg long GetNNotifyMaps();
afx_msg void SetNNotifyMaps(long nNewValue);
afx_msg long GetGlobSize();
afx_msg void SetGlobSize(long nNewValue);
afx_msg long GetRefCount();
afx_msg void SetRefCount(long nNewValue);
afx_msg BSTR GetVersion();
afx_msg void SetVersion(LPCTSTR lpszNewValue);
afx_msg BSTR GetUOM();
afx_msg void SetUOM(LPCTSTR lpszNewValue);
afx_msg BSTR GetLink();
afx_msg void SetLink(LPCTSTR lpszNewValue);
afx_msg long MMFClose();
afx_msg long MMFAddGlob(LPCTSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long
Type, long Extra);
afx_msg long MMFAddGlobEx(LPCTSTR GlobName, LPCTSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize, l
ong Type, long Extra);
afx_msg BOOL GetFirstGlob();
afx_msg BOOL GetNextGlob();
afx_msg void Erase();
afx_msg void MMFErase();
afx_msg void Insert(long value, long index);
afx_msg long ResizeMMF(long NewSize);
afx_msg void SendNotifyX(short NotifyID, short Value);
afx_msg BOOL FormatMMF(long NotifyLimit);
afx_msg long IndexOf(LPCTSTR GlobName);
afx_msg long GetValue(long Dim2, long Dim1);
afx_msg void SetValue(long Dim2, long Dim1, long nNewValue);
afx_msg long GetNotifyHandle(short index);
afx_msg void SetNotifyHandle(short index, long nNewValue);
afx_msg short GetAbValue(long index);
afx_msg void SetAbValue(long index, short nNewValue);
afx_msg long GetAiValue(long index);
afx_msg void SetAiValue(long index, long nNewValue);
afx_msg short GetAiValue(long index);
afx_msg void SetAiValue(long index, short nNewValue);
afx_msg long GetValueSD(long n);
afx_msg void SetValueSD(long n, long nNewValue);
afx_msg long GetNotifyList(long index);
afx_msg void SetNotifyList(long index, long nNewValue);
afx_msg BOOL GetNotifyOnChange(LPCTSTR GlobName);
afx_msg void SetNotifyOnChange(LPCTSTR GlobName, BOOL bNewValue);
//}}AFX_DISPATCH
DECLARE_DISPATCH_MAP()

afx_msg void AboutBox();
afx_msg long OnValueChanged(UINT LONG);

```

```

// Event maps
//{{AFX_EVENT(CGlobCtrl)
void FireChange(short PropID, short Value, long SendID)
{FireEvent(eventidChange,EVENT_PARAM(VTS_I2 VTS_I2 VTS_I4), PropID, Value, SendID);}
//}}AFX_EVENT
DECLARE_EVENT_MAP()

// Dispatch and event IDs
public:
enum {
//{{AFX_DISP_ID(CGlobCtrl)
dispidGlobIndex = 1L,
dispidDim1Size = 2L,
dispidDim2Size = 3L,
dispidElementSize = 4L,
dispidType = 5L,
dispidExtra = 6L,
dispidDataSize = 7L,
dispidGlobName = 8L,
dispidFileSize = 9L,
dispidFileName = 10L,
dispidStatus = 11L,
dispidCommand = 12L,
dispidGlobPtr = 13L,
dispidDataPtr = 14L,
dispidAvailSize = 15L,
dispidReadOnlyMMF = 16L,
dispidNotify = 17L,
dispidValue8 = 18L,
dispidValue32 = 19L,
dispidValue16 = 20L,
dispidStrValue = 21L,
dispidFullPath = 22L,
dispidAutoSendNotify = 23L,
dispidNHandles = 24L,
dispidNNotifyMaps = 25L,
dispidGlobSize = 26L,
dispidRefCount = 27L,
dispidVersion = 28L,
dispidUOM = 29L,
dispidLink = 30L,
dispidValue = 43L,
dispidCloseMMF = 31L,
dispidAddNew = 32L,
dispidAddNewEx = 33L,
dispidGetFirstGlob = 34L,
dispidGetNextGlob = 35L,
dispidErase = 36L,
dispidEraseMMF = 37L,
dispidNotifyHandle = 44L,
dispidAValue8 = 45L,
dispidAValue32 = 46L,

```

```

dispidAValue16 = 47L,
dispidValueSD = 48L,
dispidInsert = 38L,
dispidResizeMMF = 39L,
dispidSendNotify = 40L,
dispidNotifyMap = 49L,
dispidFormatMMF = 41L,
dispidIndexOf = 42L,
dispidNotifyOnChange = 50L,
eventidChange = 1L,
//}}AFX_DISP_ID
};

```

private:

```

tGlob * GlobPtr;
long datasize;
CString m_GlobName;
BOOL m_Notify;
CMutex *GlobLock;
CMutex *MMFLock;
CString m_FullPath;
CString m_FileName;
int m_FileSize;
HANDLE f;
HANDLE hFileMapT;
HANDLE s_hFileMap;
tControl *lpView;
LPBYTE lpLast;
BOOL m_AutoNotify;
long m_MaxLinks;

CString SPX_NOTIFY_MUTEX;
CString SPX_MMF_MUTEX;

```

//private member functions

```

long MMFCreat(void);
long MMFOpen(LPCTSTR, long, LPCTSTR, long);
long MMFRemapView( long );
long MMFGetGlobPtr(LPCTSTR);
tGlob *MMFNextAvailGlob( long );
tGlob *MMFFirstGlob();
void MMFEraseGlob( tGlob *);
long MMFGetGlobIx(LPCTSTR);
void MMFResetGlobs(void);
BOOL AddNotify(tGlob*,HWND);
BOOL RemoveNotify(tGlob*,HWND);
int FindHandle(HWND);
void SendNotify(tGlob*, tControl*, short, short);
void MMFClearGlobBits(long BitMap);
long SyncFileSize();
void SetBitMap(tGlob*, long index);
BOOL RemoveNotifyX(tGlob* GlobPtr, HWND my_hWnd);
long MMFClearGlobBit(tGlob* GPt, long index);

```



```
    BOOL NotifyListIsEmpty(tGlob *lpGlob);  
    long SetVisible();  
};
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_GLOBCTL_H__5F20D2E4_788C_11D1_9A9B_020701045A6B__INCLUDED)
```

Microsoft Visual Studio
C++ Compiler
Version 14.00.25123.1
Copyright (c) Microsoft Corporation
All rights reserved.
Microsoft Visual Studio
C++ Compiler
Version 14.00.25123.1
Copyright (c) Microsoft Corporation
All rights reserved.

```
// Copyright 1998, 1999 SPX Corporation
// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information
```

```
#include "stdafx.h"
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```
// Copyright 1998, 1999 SPX Corporation
```

```
#if !defined(AFX_STDAFX_H_5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_)
```

```
#define AFX_STDAFX_H_5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_
```

```
#if _MSC_VER >= 1000
```

```
#pragma once
```

```
#endif // _MSC_VER >= 1000
```

```
// stdafx.h : include file for standard system include files,
```

```
// or project specific include files that are used frequently,
```

```
// but are changed infrequently
```

```
#define VC_EXTRALEAN // Exclude rarely-used stuff from Windows headers
```

```
#include <afxctl.h> // MFC support for ActiveX Controls
```

```
// Delete the two includes below if you do not wish to use the MFC
```

```
// database classes
```

```
#include <afxdb.h> // MFC database classes
```

```
#include <afxdao.h> // MFC DAO database classes
```

```
[[[AFX_INSERT_LOCATION]]]
```

```
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_STDAFX_H_5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_)
```

```
// Copyright 1998, 1999 SPX Corporation
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Glob.rc
//
```

```
#define IDS_GLOB 1
#define IDD_ABOUTBOX_GLOB 1
#define IDB_GLOB 1
#define IDI_ABOUTDLL 1
#define IDS_GLOB_PPG 2
#define IDS_GLOB_PPG_CAPTION 200
#define IDD_PROPPAGE_GLOB 200
```

```
// Next default values for new objects
//
```

```
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 203
#define _APS_NEXT_COMMAND_VALUE 32768
#define _APS_NEXT_CONTROL_VALUE 201
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
```

```

// Copyright 1998, 1999 SPX Corporation
#ifdef AFX_GLOBPPG_H__5F20D2E6_78BC_11D1_9A9B_020701045A6B__INCLUDED_
#define AFX_GLOBPPG_H__5F20D2E6_78BC_11D1_9A9B_020701045A6B__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// GlobPpg.h : Declaration of the CGlobPropPage property page class.

////////////////////

// CGlobPropPage : See GlobPpg.cpp for implementation.

class CGlobPropPage : public COlePropertyPage
{
    DECLARE_DYNCREATE(CGlobPropPage)
    DECLARE_OLECREATE_EX(CGlobPropPage)

// Constructor
public:
    CGlobPropPage();

// Dialog Data
//{{AFX_DATA(CGlobPropPage)
enum { IDD = IDD_PROPPAGE_GLOB };
    // NOTE - ClassWizard will add data members here.
    // DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Message maps
protected:
   //{{AFX_MSG(CGlobPropPage)
    // NOTE - ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_GLOBPPG_H__5F20D2E6_78BC_11D1_9A9B_020701045A6B__INCLUDED)

```

```
// Copyright 1998, 1999 SPX Corporation
// Glob.cpp : Implementation of CGlobApp and DLL registration.
```

```
#include "stdafx.h"
#include "Glob.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
CGlobApp NEAR theApp;
```

```
const GUID CDECL BASED_CODE _tlid =
    { 0x5f20d2d3, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };
const WORD _wVerMajor = 1;
const WORD _wVerMinor = 0;
```

```
////////////////////////////////////
// CGlobApp::InitInstance - DLL initialization
```

```
BOOL CGlobApp::InitInstance()
{
    BOOL bInit = COleControlModule::InitInstance();

    if (bInit)
    {
        // TODO: Add your own module initialization code here.
    }

    return bInit;
}
```

```
////////////////////////////////////
// CGlobApp::ExitInstance - DLL termination
```

```
int CGlobApp::ExitInstance()
{
    // TODO: Add your own module termination code here.

    return COleControlModule::ExitInstance();
}
```

```
////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry
```

```
STDAPI DllRegisterServer(void)
```

```

{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleRegisterTypeLib(AfxGetInstanceHandle(), _tlid))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (IColeObjectFactoryEx::UpdateRegistryAll(TRUE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

```

```

////////////////////////////////////
// DllUnregisterServer - Removes entries from the system registry

```

```

STDAPI DllUnregisterServer(void)
{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleUnregisterTypeLib(_tlid, _wVerMajor, _wVerMinor))
        return ResultFromScode(SELFREG_E_TYPELIB);

    if (IColeObjectFactoryEx::UpdateRegistryAll(FALSE))
        return ResultFromScode(SELFREG_E_CLASS);

    return NOERROR;
}

```

```

: Copyright 1998, 1999 SPX Corporation
: Glob.def : Declares the module parameters.

```

LIBRARY "GLOB.OCX"

EXPORTS

```
DllCanUnloadNow    @1 PRIVATE
DllGetClassObject  @2 PRIVATE
DllRegisterServer  @3 PRIVATE
DllUnregisterServer @4 PRIVATE
```



```
// Copyright 1998, 1999 SPX Corporation
// Glob.odl : type library source for ActiveX Control project.

// This file will be processed by the Make Type Library (mktypelib) tool to
// produce the type library (Glob.tlb) that will become a resource in
// Glob.ocx.
```

```
#include <olectl.h>
#include <dispidids.h>
```

```
[ uuid(5F20D2D3-788C-11D1-9A9B-020701045A6B), version(1.0),
  helpfile("Glob.hlp"),
  helpstring("Glob MMF Interface"),
  control ]
```

```
library GLOBLib
```

```
{
```

```
    importlib(STDOLE_TLB);
    importlib(STDTYPE_TLB);
```

```
    // Primary dispatch interface for CGlobCtrl
```

```
    [ uuid(5F20D2D4-788C-11D1-9A9B-020701045A6B),
      helpstring("Dispatch interface for Glob Control"), hidden ]
    dispinterface _DGlob
```

```
        properties:
```

```
            // NOTE - ClassWizard will maintain property information here.
```

```
            // Use extreme caution when editing this section.
```

```
            //{AFX_ODL_PROP(CGlobCtrl)
```

```
            [id(DISPID_HWND)] OLE_HANDLE hWnd;
```

```
            [id(1)] long GlobIndex;
```

```
            [id(2)] long Dim1Size;
```

```
            [id(3)] long Dim2Size;
```

```
            [id(4)] long ElementSize;
```

```
            [id(5)] long Type;
```

```
            [id(6)] long Extra;
```

```
            [id(7)] long DataSize;
```

```
            [id(8)] BSTR GlobName;
```

```
            [id(9)] long FileSize;
```

```
            [id(10)] BSTR FileName;
```

```
            [id(11)] long Status;
```

```
            [id(12)] long Command;
```

```
            [id(13)] long GlobPtr;
```

```
            [id(14)] long DataPtr;
```

```
            [id(15)] long AvailSize;
```

```
            [id(16)] boolean ReadOnlyMMF;
```

```
            [id(17)] boolean Notify;
```

```
            [id(18)] short Value8;
```

```
            [id(19)] long Value32;
```

```
            [id(20)] short Value16;
```

```
            [id(21)] BSTR StrValue;
```

```
            [id(22)] BSTR FullPath;
```

```

[id(23)] boolean AutoSendNotify;
[id(24)] long nHandles;
[id(25)] long nNotifyMaps;
[id(26)] long GlobSize;
[id(27)] long RefCount;
[id(28)] BSTR Version;
[id(29)] BSTR UOM;
[id(30)] BSTR Link;
//}}AFX_ODL_PROP

```

methods:

```

// NOTE - ClassWizard will maintain method information here.
// Use extreme caution when editing this section.
//{{AFX_ODL_METHOD(CGlobCtrl)
[id(43), propget] long Value(long Dim2, long Dim1);
[id(43), propput] void Value(long Dim2, long Dim1, long nNewValue);
[id(31)] long CloseMMF();
[id(32)] long AddNew(BSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra);
[id(33)] long AddNewEx(BSTR GlobName, BSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra);
[id(34)] boolean GetFirstGlob();
[id(35)] boolean GetNextGlob();
[id(36)] void Erase();
[id(37)] void EraseMMF();
[id(44), propget] long NotifyHandle(short index);
[id(44), propput] void NotifyHandle(short index, long nNewValue);
[id(45), propget] short aValue8(long index);
[id(45), propput] void aValue8(long index, short nNewValue);
[id(46), propget] long aValue32(long index);
[id(46), propput] void aValue32(long index, long nNewValue);
[id(47), propget] short aValue16(long index);
[id(47), propput] void aValue16(long index, short nNewValue);
[id(48), propget] long ValueSD(long n);
[id(48), propput] void ValueSD(long n, long nNewValue);
[id(38)] void Insert(long value, long index);
[id(39)] long ResizeMMF(long NewSize);
[id(40)] void SendNotify(long NotifyID, long Value);
[id(49), propget] long NotifyMap(long index);
[id(49), propput] void NotifyMap(long index, long nNewValue);
[id(41)] boolean FormatMMF(long NotifyLimit);
[id(42)] long IndexOf(BSTR GlobName);
[id(50), propget] boolean NotifyOnChange(BSTR GlobName);
[id(50), propput] void NotifyOnChange(BSTR GlobName, boolean bNewValue);
//}}AFX_ODL_METHOD

[id(DISPID_ABOUTBOX)] void AboutBox();
};

```

// Event dispatch interface for CGlobCtrl

[uuid(5F20D2D5-788C-11D1-9A9B-020701045A6B),

```

helpstring("Event interface for Glob Control") ]
dispinterface _DGlobEvents
{
    properties:
        // Event interface has no properties

    methods:
        // NOTE - ClassWizard will maintain event information here.
        // Use extreme caution when editing this section.
        //{{AFX_ODL_EVENT(CGlobCtrl)
        [id(1)] void Change(short PropID, short Value, long SendID);
        //}}AFX_ODL_EVENT
};

// Class information for CGlobCtrl

[ uuid(5F20D2D6-788C-11D1-9A9B-020701045A6B),licensed,
  helpstring("Glob Control"), control ]
coclass Glob
{
    [default] dispinterface _DGlob;
    [default, source] dispinterface _DGlobEvents;

    //{{AFX_APPEND_ODL}}
    //}}AFX_APPEND_ODL}}
};

```

```
// Copyright 1998, 1999 SPX Corporation
// GlobCtrl.cpp : Implementation of the CGlobCtrl ActiveX Control class.
```

```
#include "stdafx.h"
#include "Glob.h"
#include "GlobCtrl.h"
#include "GlobPpg.h"
#include "sys/types.h" // for file status buffer _stat
#include "sys/stat.h" // for _fstat file status call
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
#define MIN_FILESIZE 8192
#define LOG_ERRORS FALSE
```

```
HANDLE myhWnd;
```

```
HANDLE ADhInstance;
unsigned char bResFlag;
HANDLE *iaModules;
int *iaGlobal;
BSTR saGlobal;
CString refTemp;
```

```
//BOOL LineDebug = false;
```

```
LPCTSTR m_Message = "msgGlobChange";
//long BitList[32];
```

```
// HELPER FUNCTION PROTOTYPES -RK
CString GetName(CString);
long Power(int);
void LogErrorString(CString errstr);
```

```
IMPLEMENT_DYNCREATE(CGlobCtrl, COleControl)
```

```
////////////////////////////////////
//Register for an external windows message
```

```
UINT USER_VALUECHANGED = RegisterWindowMessage(m_Message);
```

```
////////////////////////////////////
// Message map
```

```
BEGIN_MESSAGE_MAP(CGlobCtrl, COleControl)
//{{AFX_MSG_MAP(CGlobCtrl)
//}}AFX_MSG_MAP
```



```

DISP_PROPERTY_PARAM(CGlobCtrl, "Value", GetValue, SetValue, VT_I4, VTS_I4 VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyHandle", GetNotifyHandle, SetNotifyHandle, VT_I4, VTS_I2)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue8", GetAbValue, SetAbValue, VT_I2, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue32", GetAIValue, SetAIValue, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "aValue16", GetAiValue, SetAiValue, VT_I2, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "ValueSD", GetValueSD, SetValueSD, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyMap", GetNotifyList, SetNotifyList, VT_I4, VTS_I4)
DISP_PROPERTY_PARAM(CGlobCtrl, "NotifyOnChange", GetNotifyOnChange, SetNotifyOnChange, VT_BOOL, VTS_BSTR)
DISP_DEFVALUE(CGlobCtrl, "Value32")
DISP_STOCKPROP_HWND()
//}}AFX_DISPATCH_MAP
DISP_FUNCTION_ID(CGlobCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY, VTS_NONE)
END_DISPATCH_MAP()

```

```

////////////////////////////////////
// Event map

```

```

BEGIN_EVENT_MAP(CGlobCtrl, COleControl)
//{{AFX_EVENT_MAP(CGlobCtrl)
EVENT_CUSTOM("Change", FireChange, VTS_I2 VTS_I2 VTS_I4)
//}}AFX_EVENT_MAP
END_EVENT_MAP()

```

```

////////////////////////////////////
// Property pages

```

```

// TODO: Add more property pages as needed. Remember to increase the count!
BEGIN_PROPPAGEIDS(CGlobCtrl, 1)
PROPPAGEID(CGlobCtrl::guid)
END_PROPPAGEIDS(CGlobCtrl)

```

```

////////////////////////////////////
// Initialize class factory and guid

```

```

IMPLEMENT_OLECREATE_EX(CGlobCtrl, "GLOB.GlobCtrl.1",
    0x5f20d2d6, 0x788c, 0x11d1, 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b)

```

```

////////////////////////////////////
// Type library ID and version

```

```

IMPLEMENT_OLETYPELIB(CGlobCtrl, _tlid, _wVerMajor, _wVerMinor)

```

```

////////////////////////////////////
// Interface IDs

```

```

const IID BASED_CODE IID_DGlob =
    { 0x5f20d2d4, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };

```

```

const IID BASED_CODE IID_DGlobEvents =
    { 0x5f20d2d5, 0x788c, 0x11d1, { 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b } };

////////////////////////////////////
// Control type information

static const DWORD BASED_CODE _dwGlobOleMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;

IMPLEMENT_OLECTLTYPE(CGlobCtrl, IDS_GLOB, _dwGlobOleMisc)

////////////////////////////////////
// CGlobCtrl::CGlobCtrlFactory::UpdateRegistry -
// Adds or removes system registry entries for CGlobCtrl

BOOL CGlobCtrl::CGlobCtrlFactory::UpdateRegistry(BOOL bRegister)
{
    // TODO: Verify that your control follows apartment-model threading rules.
    // Refer to MFC TechNote 64 for more information.
    // If your control does not conform to the apartment-model rules, then
    // you must modify the code below, changing the 6th parameter from
    // afxRegApartmentThreading to 0.

    if (bRegister)
        return AfxOleRegisterControlClass(
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_GLOB,
            IDB_GLOB,
            afxRegApartmentThreading,
            _dwGlobOleMisc,
            _tlid,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}

////////////////////////////////////
// Licensing strings

static const TCHAR BASED_CODE _szLicFileName[] = _T("Glob.lic");

static const WCHAR BASED_CODE _szLicString[] =
    L"Copyright (c) 1999 SPX";

```

```

////////////////////////////////////
// CGlobCtrl::CGlobCtrlFactory::VerifyUserLicense -
// Checks for existence of a user license

```

```

BOOL CGlobCtrl::CGlobCtrlFactory::VerifyUserLicense()
{
    ..
    return AfxVerifyLicFile(AfxGetInstanceHandle(), _szLicFileName,
        _szLicString);
}

```

```

////////////////////////////////////
// CGlobCtrl::CGlobCtrlFactory::GetLicenseKey -
// Returns a runtime licensing key

```

```

BOOL CGlobCtrl::CGlobCtrlFactory::GetLicenseKey(DWORD dwReserved,
    BSTR FAR* pbstrKey)
{
    if (pbstrKey == NULL)
        return FALSE;

    *pbstrKey = SysAllocString(_szLicString);
    return (*pbstrKey != NULL);
}

```

```

////////////////////////////////////
// CGlobCtrl::CGlobCtrl - Constructor

```

```

CGlobCtrl::CGlobCtrl()
{
    InitializeIIDs(&IID_DGlob, &IID_DGlobEvents);

    // TODO: Initialize your control's instance data here.
    SetInitialSize( 32, 32 ); // Force to have a certain size at startup

    lpLast = NULL;
    lpView = NULL;
    s_hFileMap = NULL;
    hFileMapT = NULL;
    f = NULL;
    GlobLock = NULL;
    MMFLock = NULL;
    m_FileName = "C:\\GLOBMMF";
    m_FileSize = MIN_FILESIZE;

    GlobPtr = 0;
    m_Notify = false;
}

```



```

////////////////////////////////////
// CGlobCtrl::~CGlobCtrl - Destructor

```

```

CGlobCtrl::~CGlobCtrl()
{
    // TODO: Cleanup your control's instance data here.
    RemoveNotify(GlobPtr,GetSafeHwnd());
    if (lpView) {
        lpView->RefCount--;
        UnmapViewOfFile((LPVOID) lpView);
        GlobPtr = NULL;
        lpView=NULL;
        lpLast=NULL;
        CloseHandle(s_hFileMap);
        CloseHandle(f);
    }
    if (GlobLock) delete GlobLock;
    GlobLock = NULL;
    if (MMFLock) delete MMFLock;
    MMFLock = NULL;
}

```

```

////////////////////////////////////
// CGlobCtrl::OnDraw - Drawing function

```

```

void CGlobCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Replace the following code with your own drawing code.
    CRect r;
    CPictureHolder pict;
    if(!AmbientUserMode()) {
        r = rcBounds;
        r.right = r.left + 31;
        r.bottom = r.top + 31;
        pict.CreateFromBitmap(IDB_GLOB);
        pict.Render(pdc,r,r);
        SetControlSize(32,32);
    } else {
        ShowWindow(SW_HIDE);
    }
}

```

```

////////////////////////////////////
// CGlobCtrl::DoPropExchange - Persistence support

```

```

void CGlobCtrl::DoPropExchange(CPropExchange* pPX)

```

```

{

CString strResult;
ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
COleControl::DoPropExchange(pPX);
long ret;

// TODO: Call PX_ functions for each persistent custom property.
{

// Make FileName property Persistent
PX_String(pPX, _T("FileName"), m_FileName, "C:\\GLOBMMF");
SetFileName(m_FileName);

// make GlobName persistent
PX_String (pPX, _T("GlobName"), m_GlobName, "");
SetGlobName( m_GlobName ); // look up the Glob for this name, should relookup

PX_Bool(pPX, _T("AutoNotify"), m_AutoNotify, true);

ret = SetVisible();
}

////////////////////////////////////
// CGlobCtrl::OnResetState - Reset control to default state

void CGlobCtrl::OnResetState()
{
COleControl::OnResetState(); // Resets defaults found in DoPropExchange

// TODO: Reset any other control state here.
}

////////////////////////////////////
// CGlobCtrl::AboutBox - Display an "About" box to the user

void CGlobCtrl::AboutBox()
{
CDialog dlgAbout(IDD_ABOUTBOX_GLOB);
dlgAbout.DoModal();
}

////////////////////////////////////
// CGlobCtrl message handlers

long CGlobCtrl::GetGlobIndex()
{

```

```

// TODO: Add your property handler here
if (GlobPtr)
    return (int)(GlobPtr)-(int)lpView;
return -1;
}

void CGlobCtrl::SetGlobIndex(long nNewValue)
{
    // TODO: Add your property handler here

    if ((nNewValue >= lpView->FirstGlob) && (nNewValue <= lpView->NextAvail)) // in range?...
    {
        RemoveNotify(GlobPtr,GetSafeHwnd());
        GlobPtr = (tGlob *)((int)lpView + nNewValue); // hope caller knows what he's doing
        if (GlobPtr)
        {
            datasize = GlobPtr->datasize;
            SetGlobName((LPCTSTR)GlobPtr->name);
        }
    }
    SetModifiedFlag();
}

long CGlobCtrl::GetDim1Size()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->dim1;
    return -1;
}

void CGlobCtrl::SetDim1Size(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->dim1 = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetDim2Size()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->dim2;
    return -1;
}

void CGlobCtrl::SetDim2Size(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)

```

```

    GlobPtr->dim2 = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetElementSize()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->eltsize;
    return -1;
}

void CGlobCtrl::SetElementSize(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->eltsize = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetType()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->type;
    return -1;
}

void CGlobCtrl::SetType(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->type = (short)nNewValue;
    SetModifiedFlag();
}

long CGlobCtrl::GetExtra()
{
    // TODO: Add your property handler here

    if (GlobPtr)
        return GlobPtr->extra;
    return -1;
}

void CGlobCtrl::SetExtra(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr)
        GlobPtr->extra = (short)nNewValue;
    SetModifiedFlag();
}

```

```
long CGlobCtrl::GetDataSize()
```

```
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->datasize;
    return 0;
}
```

```
void CGlobCtrl::SetDataSize(long nNewValue)
```

```
{
    // TODO: Add your property handler here
    if (GlobPtr) {
        GlobPtr->datasize = (short)nNewValue;
        datasize = GlobPtr->datasize;
    }
    SetModifiedFlag();
}
```

```
LPCTSTR CGlobCtrl::GetGlobName()
```

```
{
    CString strResult;
    BYTE nam[17];
    int i;
    if (GlobPtr)
    {
        for (i=0;i<16;i++) nam[i] = GlobPtr->name[i];
        nam[16] = '\0';
        strResult = nam;//m_GlobName;//nam;//GlobPtr->name;
    }
    return strResult.AllocSysString();
}
```

```
void CGlobCtrl::SetGlobName(LPCTSTR lpszNewValue)
```

```
{
    CString MutexName;
    // Changing value of GlobName does read of Glob. If found, new properties are seen.
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *) MMFGetGlobPtr(lpszNewValue);
    if (GlobPtr)
    {
        datasize = GlobPtr->datasize;
        m_GlobName = lpszNewValue;
    }
    else
    {
        m_GlobName.Empty();
    }
}
```

```
SetModifiedFlag();
```

```

}

long CGlobCtrl::GetFileSize()
{
    // TODO: Add your property handler here

    return m_FileSize;
}

void CGlobCtrl::SetFileSize(long nNewValue)
{
    // TODO: Add your property handler here
    SetModifiedFlag();
}

BSTR CGlobCtrl::GetFileName()
{
    CString strResult;
    // TODO: Add your property handler here
    strResult = m_FileName;
    return strResult.AllocSysString();
}

void CGlobCtrl::SetFileName(LPCTSTR lpszNewValue)
{
    long retval;
    CString oldfilename;
    CString CurrentGlobName;

    CurrentGlobName = GetGlobName();
    oldfilename = m_FileName;
    m_FileName.Format("%s", lpszNewValue);
    m_FileSize = 8192;

    CString x;
    //AfxMessageBox("Setting file name.");
    retval = MMFCreate();

    //x.Format("MMFCreate returned: %i", retval);
    //AfxMessageBox(x);

    if (retval != OK) {
        //AfxMessageBox("Set Filename failed.");
        m_FileName = oldfilename;
        retval = MMFCreate();
        if (retval != OK) {
            m_FileName = "GlobMMF";
            MMFCreate();
            SetGlobName(CurrentGlobName);
            ThrowError(0, "failed to create MMF!", 0);
        }
        SetGlobName(CurrentGlobName);
    }
}

```

```

        ThrowError(0,"Invalid FileName.",0);
    }
    //AfxMessageBox("File Name set.");
    SetGlobName(CurrentGlobName);
    SetModifiedFlag();
}

long CGlobCtrl::GetStatus()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->status;
    return -1;
}

void CGlobCtrl::SetStatus(long nNewValue)
{
    // TODO: Add your property handler here.
    if (GlobPtr) {
        GlobPtr->status = (short)nNewValue;
        SendNotify(GlobPtr,lpView,IDSTATUS,(short)nNewValue);
    }
    SetModifiedFlag();
}

long CGlobCtrl::GetCommand()
{
    if (GlobPtr)
        return GlobPtr->command;
    return -1;
}

void CGlobCtrl::SetCommand(long nNewValue)
{
    // TODO: Add your property handler here
    if (GlobPtr) {
        GlobPtr->command = (short)nNewValue;
        SendNotify(GlobPtr,lpView,IDCOMMAND,(short)nNewValue);
    }
    SetModifiedFlag();
}

long CGlobCtrl::GetGlobPtr()
{
    // TODO: Add your property handler here

    return (long)GlobPtr;
}

void CGlobCtrl::SetGlobPtr(long nNewValue)

```

```

{
    // TODO: Add your property handler here
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *)nNewValue; // hope caller know what he's doing.
    datasize = GlobPtr->datasize;
    SetGlobName((LPCTSTR)GlobPtr->name);
    SetModifiedFlag();
}

```

```

long CGlobCtrl::GetDataPtr()

```

```

{
    // TODO: Add your property handler here

    if(GlobPtr)
        return (long)(GlobPtr->data); // + sizeof(tGlob);
    else
        return 0;
}

```

```

void CGlobCtrl::SetDataPtr(long nNewValue)

```

```

{
    // TODO: Add your property handler here

    // SetModifiedFlag();
}

```

```

long CGlobCtrl::GetAvailSize()

```

```

{
    // TODO: Add your property handler here
    if (lpView)
        return m_FileSize - lpView->NextAvail;
    else
        return 0;
}

```

```

void CGlobCtrl::SetAvailSize(long nNewValue)

```

```

{
    // TODO: Add your property handler here

    // SetModifiedFlag();
}

```

```

BOOL CGlobCtrl::GetReadOnlyMMF()

```

```

{
    // TODO: Add your property handler here
    if(lpView)
        return (BOOL)lpView->ReadOnly;
    else
        return 0;
}

```



```
void CGlobCtrl::SetReadOnlyMMF(BOOL bNewValue)
```

```
{  
    // TODO: Add your property handler here  
    lpView->ReadOnly = (int)bNewValue;  
    SetModifiedFlag();  
}
```

```
BOOL CGlobCtrl::GetNotify()
```

```
{  
    // TODO: Add your property handler here  
  
    return m_Notify;  
}
```

```
void CGlobCtrl::SetNotify(BOOL bNewValue)
```

```
{  
    // TODO: Add your property handler here
```

```
    BOOL ret;
```

```
    HWND hWind;
```

```
    CSingleLock LockMe(GlobLock);
```

```
    if(!AmbientUserMode()) {
```

```
        ThrowError(CTL_E_PERMISSIONDENIED,"This property can only be set at runtime.",0);
```

```
        return;
```

```
    }
```

```
    if(!GlobPtr) {
```

```
        ThrowError(CTL_E_PERMISSIONDENIED,"GlobName property is not set.\n Can not register Glob for notification.")
```

```
    }
```

```
    return;
```

```
    }
```

```
    LockMe.Lock(); // waits infinitely for resource to be available.
```

```
        // can use a timeout value as a parameter (ms) if desired.
```

```
    hWind = GetSafeHwnd();
```

```
    if (bNewValue)
```

```
        ret = AddNotify(GlobPtr,hWind);
```

```
    else
```

```
        ret = RemoveNotify(GlobPtr,hWind);
```

```
    if (ret)
```

```
        m_Notify = bNewValue;
```

```
    LockMe.Unlock();
```

```
    SetModifiedFlag();
```

```
}
```

```

long CGlobCtrl::GetValue(long Dim2, long Dim1)
{
    long l;
    if (GlobPtr)
    {
        l = (Dim2 * GlobPtr->dim1) + Dim1;
        if ((l * GlobPtr->elsize) < GlobPtr->datasize)
        {
            switch (GlobPtr->elsize)
            {
                case 4: return GlobPtr->data.Long[l];
                case 2: return GlobPtr->data.Short[l];
                default: return GlobPtr->data.Byte[l];
            }
        }
    }
    return -1;
}

```

```

void CGlobCtrl::SetValue(long Dim2, long Dim1, long nNewValue)
{
    long l;
    if (!lpView->ReadOnly))
    {
        if (GlobPtr)
        {
            l = (Dim2 * GlobPtr->dim1) + Dim1;
            if ((l * GlobPtr->elsize) < GlobPtr->datasize)
            {
                switch (GlobPtr->elsize)
                {
                    case 4: GlobPtr->data.Long[l] = nNewValue;
                        break;
                    case 2: GlobPtr->data.Short[l] = (short)nNewValue;
                        break;
                    default: GlobPtr->data.Byte[l] = (BYTE) nNewValue;
                        break;
                }
                // notify controls on list of change
                //if(GlobPtr->notify !=0)
                SendNotify(GlobPtr,lpView,IDVALUE,0);
            }
        }
        //SetModifiedFlag();
    }
}

```

```

/*****
MMFCreate: Create file (or just open it) and map a View to it
*****/
long CGlobCtrl::MMFCreate(void)
{

```

```

struct _stat buf;
int result,i;
CString MMFName;
BOOL FirstMapping;
char buffer[256];
//CString temp;
//BYTE *testView;
//BYTE testRead;
BYTE *MMFLastByte;
long OldFileSize;
long NewFileArea;
long HandleListSize;
DWORD fileretval;
long errcode;

if (lpView){
    //reftemp.Format("RefCount-- (MMFCreate) f=%i",f);
    //AfxMessageBox(reftemp);
    lpView->RefCount--;
    UnmapViewOfFile((LPVOID) lpView);
    GlobPtr = NULL;
    lpView=NULL;
    lpLast=NULL;
    CloseHandle(s_hFileMap);
    CloseHandle(f);
    if(GlobLock) delete GlobLock;
    GlobLock = NULL;
    if(MMFlock) delete MMFlock;
    MMFlock = NULL;

//AfxMessageBox("Made it past unmapping stuff");

fileretval = ::GetFullPathName(m_FileName,254,buffer,NULL);

if (fileretval == 0){
    m_FullPath = m_FileName;
} else {
    m_FullPath.Format("%s",(LPCTSTR)buffer);
}

//AfxMessageBox(m_FullPath);

MMFName = GetName(m_FullPath);

// Initialize the Mutex objects
SPX_NOTIFY_MUTEX = MMFName + "NOTIFY";
SPX_MMF_MUTEX = MMFName + "MMF";
GlobLock = new CMutex(false,SPX_NOTIFY_MUTEX,NULL);

```

```

MMFLock = new CMutex(false,SPX_MMF_MUTEX,NULL);

//AfxMessageBox(MMFName);
/* Get data associated with file */
result = _stat( m_FullPath, &buf ); // result will be -1 if file does not exist

//if the file exists then get its filesize
if (!result) {
    if (buf.st_size > m_FileSize)
        m_FileSize = buf.st_size; // get size of file
    OldFileSize = buf.st_size;
} else {
    OldFileSize = 0;
}

//DEBUG
//temp.Format("MMF FileSize = %d\n",m_FileSize);
//LogErrorString(temp);

// Create an in-memory memory-mapped file.
f = CreateFile( m_FullPath,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, //FileAttr,
    NULL);
if (f==INVALID_HANDLE_VALUE)
{
    //AfxMessageBox("Cant open file");
    //RaiseException(997,0,0,0);
    //ThrowError(0,"Invalid File Name",0); //ThrowError only works in properties and methods
    return ERR_INVALIDFILENAME;
}

//AfxMessageBox("File opened OK");
//Grow the file
SetFilePointer(f,m_FileSize,NULL,FILE_BEGIN);
SetEndOfFile(f);
//Error checking?

s_hFileMap = CreateFileMapping(f, NULL, PAGE_READWRITE, 0, 0, MMFName /*ViewName*/);
// s_hFileMap = CreateFileMapping((HANDLE) 0xFFFFFFFF, NULL, PAGE_READWRITE, 0, MMFSize, MMFName);

errcode = GetLastError();
//reftemp.Format("Error code %i (%i)",errcode,ERROR_ALREADY_EXISTS);
//AfxMessageBox(reftemp);
if (errcode == ERROR_ALREADY_EXISTS) {
    FirstMapping = false;
} else {

```

```

FirstMapping = true;
}

if (s_hFileMap != NULL)
{
// if (GetLastError() == ERROR_ALREADY_EXISTS) MessageBox(myhWnd, __TEXT("MMF Already Exists."), NULL, MB_
// File mapping created successfully. Map a view of the file into the address space.
lpView = (tControl *)MapViewOfFile(s_hFileMap, FILE_MAP_WRITE | FILE_MAP_READ, 0, 0, 0);
if (lpView != NULL)
{
//fill in 0's if file is expanded
if (m_FileSize > OldFileSize){
//LogErrorString("Filling in new memory with 0's\n");
MMFLastByte = (BYTE *)((long)lpView + OldFileSize);
NewFileArea = m_FileSize - OldFileSize;
memset(MMFLastByte, 0, NewFileArea);
}

//set rest of Glob Attributes
if (lpView->nNotifyMaps == 0)
lpView->nNotifyMaps = DEF_NOTIFYMAPS;
HandleListSize = 32*sizeof(long)*lpView->nNotifyMaps;
m_MaxLinks = lpView->nNotifyMaps*32;
lpView->Size = m_FileSize; // make global
lpView->FirstGlob = sizeof(tControl) + HandleListSize;
lpLast = (BYTE *)lpView + m_FileSize - sizeof(tGlob) - sizeof(int);
if (!lpView->NextAvail)
lpView->NextAvail = sizeof(tControl) + HandleListSize;

//reftemp.Format("RefCount++ (MMFCreate) f=%i",f);
//AfxMessageBox(reftemp);
lpView->RefCount++;

// clear out Notify array if this is the first mapping
if (FirstMapping) {
//AfxMessageBox("First Mapping, Clearing Notify Handles");
for (i=0; i<m_MaxLinks; i++)
lpView->NotifyHandle[i] = 0;
lpView->RefCount = 1;
// Reset globs
MMFResetGlobs();
}
return OK;
}
else
{
return ERR_CANT_MAP_VIEW_OF_FILE;
}
}
else
{

```

```

        return ERR_CANT_CREATE_FILE_MAPPING :-2;
    }

    return ERR_INVALIDFILENAME;
}

////////////////////////////////////
// Create a unique MMF view name from the MMF filename -RK
////////////////////////////////////
CString GetName(CString s)
{
    int i,j;
    CString Buffer;
    int len;

    //AfxMessageBox(s);
    j = 0;
    len = s.GetLength();

    // load Buffer with spaces
    for (i=0;i<len;i++)
        Buffer = Buffer + " ";

    for (i=0;i<len;i++) {
        if (s[i] != '\\') {
            Buffer.SetAt(j,s[i]);
            j++;
        }
    }

    Buffer.TrimRight();

    // Buffer = s.Right((len-i)-1);
    Buffer.MakeUpper();
    return Buffer;
}

long CGlobCtrl::MMFOpen(LPCTSTR FileName, long FileAttr, LPCTSTR ViewName, long FileSize)
{
    // Create an in-memory memory-mapped file
    //
    FileAttr = FILE_ATTRIBUTE_NORMAL; // force it for now
    f = CreateFile( FileName,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_ALWAYS,
        FileAttr,
        NULL);
    if (f==INVALID_HANDLE_VALUE)
    {
        // MessageBox(myhWnd,TEXT("Cant open file"),NULL,MB_OK);
        return 0;
    }
}

```

```

    }
    s_hFileMap = CreateFileMapping(f, NULL, PAGE_READWRITE, 0, FileSize, ViewName);
    // for memory-only, use "CreateFileMapping((HANDLE) 0xFFFFFFFF, .....

    if (s_hFileMap != NULL)
    {
        // if (GetLastError() == ERROR_ALREADY_EXISTS) MessageBox(myhWnd, __TEXT("MMF Already Exists."), NULL, MB_
        // File mapping created successfully. Map a view of the file into the address space.
        lpView = (tControl *)MapViewOfFile(s_hFileMap, FILE_MAP_READ | FILE_MAP_WRITE, 0, 0, 0);
        if (lpView != NULL)
        {
            // View mapped successfully.
            // To unmap the view: (This protects the data from wayward pointers). "UnmapViewOfFile((LPVOID) lpView);"
            lpView->Size = FileSize; // make global
            lpView->FirstGlob = sizeof( tControl );
            lpLast = (BYTE *)lpView + FileSize - sizeof(tGlob) - sizeof(int);
            if (!lpView->NextAvail) lpView->NextAvail = sizeof( tControl );
            return OK;
        }
    }
    else
    {
        RaiseException(999,0,0,0);
        //MessageBox(myhWnd, __TEXT("Can't map view of file."), NULL, MB_OK);
        return ERR_CANT_MAP_VIEW_OF_FILE;
    }
}
else
{
    RaiseException(998,0,0,0);
    //MessageBox(myhWnd, __TEXT("Can't create file mapping."),NULL, MB_OK);
    return ERR_CANT_CREATE_FILE_MAPPING ;-2;
}

return 0;
}

```

```

/*****
MMFRemapView: Close view and reopen as different size
*****/
long CGlobCtrl::MMFRemapView( long newsize )
{
    return MMFCreate(); // 03/12/98 RBK
}

```

```

/*****
MMFGetGlobIx: Returns index (offset) of Glob if name is found, otherwise 0
*****/
long CGlobCtrl::MMFGetGlobIx(LPCTSTR GlobName)
{
    tGlob *lpGlob;
    CString name;
}

```

```

if (GlobName != NULL)
    name = GlobName;
else
    name = "";

if (name.GetLength() == 0)
    return 0;

for (
    lpGlob=(tGlob *)((int)lpView + lpView->FirstGlob);
    lpGlob->size && (int)lpGlob<(int)lpLast;
    lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
)
    if (!_strnicmp( (const char *)GlobName, (const char *)((tGlob *)lpGlob)->name, Glob_NAME_LENGTH ))
    {
        return (int)lpGlob - (int)lpView;
    }
return 0;
}

/*****
MMFEraseGlob: Clear the Glob to zeroes, but leaving its space still linked in.
*****/
void CGlobCtrl::MMFEraseGlob( tGlob *B )
{
    int save_size;
    //CString DebugStr;

    if (B)
    {
        save_size = B->size;

        //DebugStr.Format("SaveSize = %i\n",save_size);
        //LogErrorString(DebugStr);

        memset( (BYTE *)B, 0, save_size); // clear all
        B->size = save_size; // restore size for linking past
    }
}

/*****
MMFFirstGlob: Returns pointer to 1st Glob in the MMF
*****/
tGlob *CGlobCtrl::MMFFirstGlob()
{
    return (tGlob *)((int)lpView + lpView->FirstGlob);
}

/*****
MMFNextAvailGlob: Scans from beginning for empty Glob of adequate size. If none, uses NextAvail pointer
*****/
tGlob *CGlobCtrl::MMFNextAvailGlob( long size )
{
    CString DebugStr;

```



```

int lastchance=(int)lpView + m_FileSize - size - 1;
tGlob *lpGlob;

//DEBUG
//lpGlob = MMFFirstGlob();
//DebugStr.Format("FirstGlobSize: %i\n",lpGlob->size);
//LogErrorString(DebugStr);
//END DEBUG

for ( // chain thru Glob's in MMF
    lpGlob=MMFFirstGlob(); // first Glob...
    lpGlob->size && ((int)lpGlob < lastchance); // if size is nz, within range
    lpGlob = (tGlob*)((int)lpGlob + lpGlob->size)
){
    //DebugStr.Format("Name: %s size: %i\n", lpGlob->name,lpGlob->size);
    //LogErrorString(DebugStr);
    if (lpGlob->name[0] == '\0' && lpGlob->size >= size) // if has zeroed-out name and is big enough...
        return lpGlob; // return pointer to it
}
// falls thru loop...no empties found
lpGlob = (tGlob*)((int)lpView + lpView->NextAvail); // else get pointer to next one in MMF
if ((int)lpGlob > lastchance) // is there room for it?
    lpGlob = 0; // no, return 0
return lpGlob;
}

/*****
TOOL: MMFGetGlobPtr: Given the GlobName, find and return pointer to Glob, else zero
*****/
long CGlobCtrl::MMFGetGlobPtr(LPCTSTR GlobName)
{
    tGlob *lpGlob;
    //CString temp;

    for (
        lpGlob=MMFFirstGlob();
        lpGlob->size && ((int)lpGlob < (int)lpLast);
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
    ){
        /*
        if (LineDebug) {
            temp.Format("GlobName: %s\nlpGlob = %i , lpLast = %i , Size = %i\n",((const char *)((tGlob *)lpGlob)
->name),lpGlob,lpLast,lpGlob->size);
            LogErrorString(temp);
        }
        */
        if (_strnicmp( GlobName, (const char *)((tGlob *)lpGlob)->name, Glob_NAME_LENGTH )) {
            //LineDebug = false;
            return (int)lpGlob;//true 32-bit pointer
        }
    }
}
//LineDebug = false;

```

```
return 0;
```

```
}
```

```
void CGlobCtrl::MMFResetGlobs()
```

```
{
```

```
    tGlob *lpGlob;
```

```
    int i;
```

```
    BYTE* temp;
```

```
    long* MapPtr;
```

```
    long datacount;
```

```
    for (
```

```
        lpGlob=MMFFirstGlob();
```

```
        lpGlob->size && (int)lpGlob<(int)lpLast;
```

```
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
```

```
    ){
```

```
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
```

```
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
```

```
        for (i=0;i<lpView->nNotifyMaps;i++) {
```

```
            // lpGlob->notifymap[i] = 0;
```

```
            MapPtr = (long*)temp + i;
```

```
            *MapPtr = 0;
```

```
        }
```

```
        lpGlob->command = 0;
```

```
        lpGlob->status = 0;
```

```
    }
```

```
    return;
```

```
}
```

```
void CGlobCtrl::MMFClearGlobBits(long index)
```

```
{
```

```
    tGlob *lpGlob;
```

```
    long mapIndex,bitIndex;
```

```
    BYTE* temp;
```

```
    long* MapPtr;
```

```
    long datacount;
```

```
    mapIndex = index/32;
```

```
    bitIndex = index - mapIndex*32;
```

```
    for (
```

```
        lpGlob=MMFFirstGlob();
```

```
        ((int)lpGlob <= (int)lpLast) && lpGlob->size;
```

```
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
```

```
    ){
```

```
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
```

```
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
```

```
        MapPtr = (long*)temp + mapIndex;
```

```
        //lpGlob->notifymap[mapIndex] &= (NOTIFYMASK ^ BitList[bitIndex]);
```

```
        *MapPtr &= (NOTIFYMASK ^ (1<<bitIndex));
```

```
        if(NotifyListIsEmpty(lpGlob)) lpGlob->ptrMap = 0;
```

```

    }
    return;
}

long CGlobCtrl::MMFClearGlobBit(tGlob* GPtr, long index)
{
    tGlob *lpGlob;
    long mapIndex, bitIndex;
    BYTE* temp;
    long* MapPtr;
    long count;
    long datacount;

    count = 0;
    mapIndex = index/32;
    bitIndex = index - mapIndex*32;

    for (
        lpGlob=MMFFirstGlob();
        ((int)lpGlob <= (int)lpLast) && lpGlob->size;
        lpGlob = (tGlob*)((int)lpGlob+lpGlob->size)
    ){
        datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
        temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
        MapPtr = (long*)temp + mapIndex;
        //lpGlob->notifymap[mapIndex] &= (NOTIFYMASK ^ BitList[bitIndex]);
        if(*MapPtr & (1<<bitIndex)){
            count++;
            if(GPtr == lpGlob){
                *MapPtr &= (NOTIFYMASK ^ (1<<bitIndex));
                count--;
            }
            if(NotifyListIsEmpty(lpGlob)) lpGlob->ptrMap = 0;
        }
    }
    return count;
}

```

```

BOOL CGlobCtrl::NotifyListIsEmpty(tGlob *lpGlob)
{
    BYTE* temp;
    long* MapPtr;
    int i;
    BOOL isEmpty;
    long datacount;

    isEmpty = true;
    datacount = lpGlob->dim1 * lpGlob->dim2 * lpGlob->eltsize;
    temp = (BYTE*)lpGlob + sizeof(tGlob) + datacount;
    MapPtr = (long*)temp;

    for (i=0; i<lpView->nNotifyMaps; i++){

```

```

        if(*(MapPtr+i)) isEmpty = false;
    }

    return isEmpty;
}

void CGlobCtrl::SetBitMap(tGlob* GlobPtr, long index)
{
    long mapIndex, bitIndex;
    long* MapPtr;
    BYTE *temp;
    long datacount;

    mapIndex = index/32;
    bitIndex = index - mapIndex*32;

    datacount = GlobPtr->dim1 * GlobPtr->dim2 * GlobPtr->eltsize;
    if (GlobPtr->ptrMap == 0) {
        GlobPtr->ptrMap = sizeof(tGlob) + datacount;
    }
    //GlobPtr->notifymap[mapIndex] |= BitList[bitIndex];

    temp = (BYTE*)GlobPtr + GlobPtr->ptrMap;
    MapPtr = (long*)temp + mapIndex;
    *MapPtr |= (1<<bitIndex); //BitList[bitIndex];
}

```

```

long CGlobCtrl::MMFClose()
{
    // TODO: Add your dispatch handler code here
    //reftemp.Format("RefCount-- (MMFClose) f=%i",f);
    //AfxMessageBox(reftemp);
    lpView->RefCount--;
    RemoveNotify(GlobPtr, GetSafeHwnd());
    UnmapViewOfFile((LPVOID) lpView);
    lpView=NULL;
    lpLast=NULL;
    CloseHandle(s_hFileMap);
    return CloseHandle(f);
}

```

```

long CGlobCtrl::MMFAddGlob(LPCTSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra)
{
    int count, ThisGlobSize, LeftOverCount, i;
    long bitmaploc;
    BYTE* temp;
    //CString DebugStr;
}

```

```

long* MapPtr;

RemoveNotify(GlobPtr,GetSafeHwnd());

if (!Dim2Size)                // if zero, make = 1
    Dim2Size = 1;
if (!Dim1Size)
    Dim1Size = 1;                ..

//DebugStr.Format("Looking for previous Glob: %s\n",GlobName);
//LogErrorString(DebugStr);

GlobPtr = (tGlob *)MMFGetGlobPtr( GlobName); // first look for one by this name

//DebugStr.Format("Prev Glob Ptr: %i\n",GlobPtr);
//LogErrorString(DebugStr);

if (GlobPtr)                   // found a previous, maybe can reuse space
    MMFEraseGlob( GlobPtr);    // clear it, maybe NextAvail can reuse it
count = Dim2Size * Dim1Size * ElementSize; // calc data size
ThisGlobSize = (sizeof(tGlob) + count + 3) & ~3; // add size of Glob and put on even 4-byte boundary
bitmaploc = ThisGlobSize;
ThisGlobSize += sizeof(long) * lpView->nNotifyMaps;

//DebugStr = "Looking for space for Glob\n";
//LogErrorString(DebugStr);

GlobPtr = MMFNextAvailGlob( ThisGlobSize); // get pointer to area of adequate size

//DebugStr.Format("Found Space at: %i\n", (long)GlobPtr);
//LogErrorString(DebugStr);

while (!GlobPtr)               // no room, make bigger
{
    if(lpView->RefCount > 1) {
        ThrowError(CTL_E_PERMISSIONDENIED, "MMF can not be expanded. Too many connections. \nClose all other appli
cations and try again.");
        return NULL;
    }

    //Debug *****
    //DebugStr.Format("Resizing for %s\n",GlobName);
    //LogErrorString(DebugStr);
    //DebugStr.Format("FileSize Before = %i\n",m_FileSize);
    //LogErrorString(DebugStr);
    // *****

    //LineDebug = true;
    m_FileSize += (ThisGlobSize + 4095); // calc new size of file
    m_FileSize &= ~4095;                // make size a multiple of 4096

    //Debug *****

```

```

//DebugStr.Format("FileSize After = %i\n",m_FileSize);
//LogErrorString(DebugStr);
//*****

MMFCreate(); // unmap/remap view to increase size
GlobPtr = MMFNextAvailGlob( ThisGlobSize ); // get pointer to area of adequate size

//Debug ***** ..
//DebugStr.Format("GlobPtr After Remap: %i\n",GlobPtr);
//LogErrorString(DebugStr);
//*****

SetModifiedFlag(); // properties have changed
}
// setup member variables
datasize = count;
// setup Glob data variables
LeftOverCount = GlobPtr->size - ThisGlobSize; // subtract this size from size that might have been in
GlobPtr->size = ThisGlobSize;
GlobPtr->dim2 = (short)Dim2Size;
GlobPtr->dim1 = (short)Dim1Size;
GlobPtr->eltsize = (short)ElementSize;
GlobPtr->type = (short)Type;
GlobPtr->UOM = UnitsIndex;
GlobPtr->extra = (short)Extra;
GlobPtr->command = 0;
GlobPtr->status = 0;
GlobPtr->ptrMap = 0; //bitmaploc;
temp = (BYTE*)GlobPtr + bitmaploc;
for (i=0;i<lpView->nNotifyMaps;i++) {
    //GlobPtr->notifymap[i] = 0;
    MapPtr = (long*)temp + i;
    *MapPtr = 0;
}
GlobPtr->datasize= count;
memset( GlobPtr->name, 0, Glob_NAME_LENGTH ); // clear name to zeroes
strncpy( (char *)GlobPtr->name, GlobName, Glob_NAME_LENGTH ); // copy name in
if ((int)GlobPtr == ((int)lpView + lpView->NextAvail)) // if new Glob is at end of file (not reusing of
her area)
    lpView->NextAvail += ThisGlobSize; // incr nextavail pointer
else // if bytes left over, make new [size] header for empty space left beyond this Glob
{
    if (LeftOverCount > (int)(sizeof(tGlob) + sizeof(long) * lpView->nNotifyMaps)) {
        ((tGlob *)((int)GlobPtr + ThisGlobSize))->size = LeftOverCount; // put a [size] value past this Glob to
reclaim space beyond
    } else {
        GlobPtr->size = ThisGlobSize + LeftOverCount;
    }
}

//DebugStr = "Glob successfully inserted.\n";
//LogErrorString(DebugStr);

```

```

    SetModifiedFlag();
    return (int)GlobPtr; // returns Glob pointer
}

long CGlobCtrl::MMFAddGlobEx(LPCTSTR GlobName, LPCTSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize, long Type, long Extra)
{
    int status,uom;
    CString Units;
    int count;
    //CString temp;

    count = Dim2Size * Dim1Size * ElementSize;
    if (UnitsName != NULL)
        Units = UnitsName;
    else
        Units = "";

    if (Units.GetLength() != 0) {
        uom = MMFGetGlobIx( Units );
        if (!uom) MMFAddGlob( Units, 0,0,0,0,-1,0); // add unit of measure first
        uom = MMFGetGlobIx( Units );
    } else {
        uom = 0;
    }

    status = MMFAddGlob( GlobName, uom, Dim2Size, Dim1Size, ElementSize, Type, Extra );

    //temp.Format("(%i) AddGlobEx finished successfully for: %s\n",status,GlobName);
    //LogErrorString(temp);

    return status;
}

BOOL CGlobCtrl::GetFirstGlob()
{
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *)((int)lpView + lpView->FirstGlob);
    // HG 980423 SetGlobName((LPCTSTR)GlobPtr->name);
    datasize = GlobPtr->datasize; // HG 980423 copied from SetGlobName
    m_GlobName = GlobPtr->name; // HG 980423 copied from SetGlobName
    SetModifiedFlag(); // cause properties to re-read
    if (GlobPtr->size)
        return true;
    return false;
}

BOOL CGlobCtrl::GetNextGlob()
{
    RemoveNotify(GlobPtr,GetSafeHwnd());
    GlobPtr = (tGlob *)((int)GlobPtr + GlobPtr->size);
    // HG 980423 SetGlobName((LPCTSTR)GlobPtr->name);

```

```

datasize = GlobPtr->datasize; // HG 980423 copied from SetGlobName
m_GlobName = GlobPtr->name; // HG 980423 copied from SetGlobName
SetModifiedFlag(); // cause properties to re-read
if (GlobPtr->size)
    return true;
return false;
}

void CGlobCtrl::Erase()
{
    CSingleLock LockMe(MMFlock);

    LockMe.Lock();
    RemoveNotify(GlobPtr, GetSafeHwnd());
    MMFEraseGlob( GlobPtr );
    LockMe.Unlock();
}

void CGlobCtrl::MMFErase()
{
    CSingleLock LockMe(MMFlock);

    LockMe.Lock();
    if (lpView)
    {
        int size;
        int nmaps;

        RemoveNotify(GlobPtr, GetSafeHwnd());
        size = lpView->Size;
        nmaps = lpView->nNotifyMaps;
        memset( lpView, 0, size );
        lpView->nNotifyMaps = nmaps;
        lpView->NextAvail = lpView->FirstGlob = sizeof( tControl ) + 32*sizeof(long)*nmaps;
        lpView->Size = size;
    }
    LockMe.Unlock();
}

long CGlobCtrl::GetNotifyList(long index)
{
    BYTE* temp;
    long* MapPtr;

    if (lpView) {
        if((index < 0) || (index >= lpView->nNotifyMaps)) return 0;
        // TODO: Add your property handler here
        if (GlobPtr && GlobPtr->ptrMap) {
            temp = (BYTE*)GlobPtr + GlobPtr->ptrMap;
            MapPtr = (long*)temp + index;
        }
    }
}

```



```

        return *MapPtr;
    }
}

return 0;
}

void CGlobCtrl::SetNotifyList(long index, long nblewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

BOOL CGlobCtrl::AddNotify(tGlob* GlobPtr,HWND my_hWnd)
{
    int index;

    // add my hWnd to notify list
    if(GlobPtr){
        index = FindHandle(my_hWnd); // look for a previous entry
        if (index != -1){ // if we find one, dont add another!
            SetBitMap(GlobPtr,index); // RK 042498
            return true;
        }

        // didnt find one so make one
        index = FindHandle(0); // look for first 0 entry
        if (index != -1){ // make sure there is one available
            lpView->NotifyHandle[index] = my_hWnd;
            m_Notify = true;
        }
        /*
        CString Temp;
        Temp.Format("index = %d Power(index) = %d",index,Power(index));
        AfxMessageBox(Temp);
        */

        SetBitMap(GlobPtr,index); // add ref to notify list
        return true;
    }
}

ThrowError(CTL_E_OUTOFMEMORY,"Out of memory in MMF. Can not register Glob for notification.");
return false;
}

BOOL CGlobCtrl::RemoveNotify(tGlob* GlobPtr, HWND my_hWnd)
{
    int index;

    // remove my hWnd from the notify list
    if(GlobPtr){
        m_Notify = false; // HG 980423 clear notify flag in any case
        index = FindHandle(my_hWnd); // look for handle in list
    }
}

```

```

    if (index == -1)          // not there!
        return true;        // dont need to remove anything

    lpView->NotifyHandle[index] = 0; // remove entry from list
    MMFClearGlobBits(index);
    //GlobPtr->notify &= (NOTIFYMASK ^ BitList[index]); // remove ref from notify map
    return true;
}
return false;
}

```

```

BOOL CGlobCtrl::RemoveNotifyX(tGlob* GPtr, HWND my_hWnd)
{

```

```

    int index;
    long count;

    // remove my hWnd from the notify list
    if(GlobPtr) {
        index = FindHandle(my_hWnd); // look for handle in list
        if (index == -1)             // not there!
            return true;            // dont need to remove anything
        count = MMFClearGlobBit(GPtr,index);
        if (count == 0) {
            lpView->NotifyHandle[index] = 0;
            m_Notify = false;
        }
    }
    return true;
}
return false;
}

```

```

int CGlobCtrl::FindHandle(HWND my_hWnd)
{

```

```

    int i;

    for (i=0; i<m_MaxLinks; i++)
        if (lpView->NotifyHandle[i] == my_hWnd) return i;

    return -1;
}

```

```

long CGlobCtrl::GetNotifyHandle(short index)
{

```

```

    // TODO: Add your property handler here
    if ((lpView) && (index < m_MaxLinks) && (index >= 0))
        return (long)lpView->NotifyHandle[index];

    return -1;
}

```

```

void CGlobCtrl::SetNotifyHandle(short index, long nNewValue)

```

```

{
    // TODO: Add your property handler here

    SetModifiedFlag();
}
long Power(int operand)
{
    int i;
    long value;

    if (operand == 0) {
        value = 1;
    } else {
        value = 1;
        for(i=1;i<=operand;i++)
            value *=2;
    }
    return value;
}

void CGlobCtrl::SendNotify(tGlob* GlobPtr, tControl* lpView, short IDProp, short PropValue)
{
    // TODO: Add your dispatch handler code here
    int i, mapIndex, bitIndex;
    int results;
    HWND mHwnd;
    tGlobMsg msg;
    long* tempmsg;
    long GlobID;
    int count;
    long map;

    count = 0;

    // set up message to send for notify
    msg.PropID = IDProp;
    msg.Value = PropValue;
    tempmsg = (long*)&msg;
    GlobID = (long)GlobPtr - (long)lpView;

    mHwnd = GetSafeHwnd();

    if(m_AutoNotify) {
        if(GlobPtr && GlobPtr->ptrMap) {
            for (mapIndex=0; mapIndex<lpView->nNotifyMaps; mapIndex++) {
                map = GetNotifyList(mapIndex);
                if (map != 0) count++;
                for (bitIndex = 0; bitIndex<32; bitIndex++) {
                    i = mapIndex*32+bitIndex;
                    if((map & (1<<bitIndex)) && (mHwnd != lpView->NotifyHandle[i])) {
                        results = ::PostMessage(lpView->NotifyHandle[i], USER_VALUECHANGED, *tempmsg, GlobID);
                        if (!results) // if the handle is invalid then remove it from the list

```

```

        RemoveNotify(GlobPtr,lpView->NotifyHandle[i]);
    }
}
}
if (count == 0) GlobPtr->ptrMap = 0;
}
}
}

void CGlobCtrl::OnFinalRelease()
{
    // TODO: Add your specialized code here and/or call the base class
    RemoveNotify(GlobPtr,GetSafeHwnd());
    COleControl::OnFinalRelease();
}

long CGlobCtrl::OnValueChanged(UINT lParam, LONG rParam)
{
    //unpack lParam for PropID and Value
    tGlobMsg* msg;

    msg = (tGlobMsg*)&lParam;

    FireChange(msg->PropID,msg->Value,rParam);
    return 0;
}

short CGlobCtrl::GetByteValue()
{
    // TODO: Add your property handler here
    // returns a byte (short was the only option in the wizard ;)
    if (GlobPtr)
        return GlobPtr->data.Byte[0];
    return 0;
}

void CGlobCtrl::SetByteValue(short nNewValue)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && !(lpView->ReadOnly)) {
        GlobPtr->data.Byte[0] = (BYTE)nNewValue;

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
        SendNotify(GlobPtr,lpView,IDVALUE,0);
    }
    SetModifiedFlag();
}

short CGlobCtrl::GetAbValue(long index)
{
    // TODO: Add your property handler here

```

```

    if ((GlobPtr) && (index < GlobPtr->datasize))
        return GlobPtr->data.Byte[index];
    return 0;
}

void CGlobCtrl::SetAbValue(long index, short nNewValue)
{
    // TODO: Add your property handler here ..
    if (!(lpView->ReadOnly))
        if ((GlobPtr) && (index < GlobPtr->datasize)) {
            GlobPtr->data.Byte[index] = (BYTE)nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
        }
    //SetModifiedFlag(); -RK not needed for non persistent properties
}

long CGlobCtrl::GetLValue()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->data.Long[0];
    return 0;
}

void CGlobCtrl::SetLValue(long nNewValue)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && !(lpView->ReadOnly)) {
        if (GlobPtr->elsize == 4)
            GlobPtr->data.Long[0] = nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
    }
    //SetModifiedFlag();
}

long CGlobCtrl::GetAlValue(long index)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && ((index * sizeof(long)) < GlobPtr->datasize))
        return GlobPtr->data.Long[index];
    return 0;
}

void CGlobCtrl::SetAlValue(long index, long nNewValue)
{
    // TODO: Add your property handler here

```

```

    if (!lpView->ReadOnly))
        if ((GlobPtr) && ((index * sizeof(long)) < GlobPtr->datasize)) {
            GlobPtr->data.Long[index] = nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
                SendNotify(GlobPtr,lpView,IDVALUE,0);
        }
        //SetModifiedFlag();
}

short CGlobCtrl::GetIValue()
{
    // TODO: Add your property handler here
    if (GlobPtr)
        return GlobPtr->data.Short[0];
    return 0;
}

void CGlobCtrl::SetIValue(short nNewValue)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && !lpView->ReadOnly) {
        if (GlobPtr->elsize >= 2)
            GlobPtr->data.Short[0] = nNewValue;

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);
        //SetModifiedFlag();
    }
}

short CGlobCtrl::GetAiValue(long index)
{
    // TODO: Add your property handler here
    if ((GlobPtr) && ((index * sizeof(short)) < GlobPtr->datasize))
        return GlobPtr->data.Short[index];
    return 0;
}

void CGlobCtrl::SetAiValue(long index, short nNewValue)
{
    // TODO: Add your property handler here
    if (!lpView->ReadOnly)
        if ((GlobPtr) && ((index * sizeof(short)) < GlobPtr->datasize)) {
            GlobPtr->data.Short[index] = nNewValue;

            // notify controls on list of change
            //if(GlobPtr->notify !=0)
                SendNotify(GlobPtr,lpView,IDVALUE,0);
        }
}

```

```

    //SetModifiedFlag();
}

long CGlobCtrl::GetValueSD(long n)
{
    // TODO: Add your property handler here
    if (GlobPtr)
    {
        if ((n * GlobPtr->eltsize) < GlobPtr->datasize)
        {
            switch (GlobPtr->eltsize)
            {
                case 4: return GlobPtr->data.Long[n];
                case 2: return GlobPtr->data.Short[n];
                default: return GlobPtr->data.Byte[n];
            }
        }
    }
    return -1;
}

void CGlobCtrl::SetValueSD(long n, long nNewValue)
{
    // TODO: Add your property handler here
    if (!(lpView->ReadOnly))
    {
        if (GlobPtr)
        {
            if ((n * GlobPtr->eltsize) < GlobPtr->datasize)
            {
                switch (GlobPtr->eltsize)
                {
                    case 4: GlobPtr->data.Long[n] = nNewValue;
                        break;
                    case 2: GlobPtr->data.Short[n] = (short)nNewValue;
                        break;
                    default: GlobPtr->data.Byte[n] = (BYTE) nNewValue;
                        break;
                }
                // notify controls on list of change
                //if(GlobPtr->notify !=0)
                SendNotify(GlobPtr,lpView,IDVALUE,0);
            }
        }
    }
    //SetModifiedFlag();
}

BSTR CGlobCtrl::GetStrValue()
{
    CString strResult;
    if (GlobPtr)

```

```

    strResult = (GlobPtr->data.Byte);
    // TODO: Add your property handler here

    return strResult.AllocSysString();
}

void CGlobCtrl::SetStrValue(LPCTSTR lpszNewValue)
{
    // TODO: Add your property handler here
    //CString strResult(lpszNewValue);
    int i;
    int size;

    if (!(lpView->ReadOnly) && (GlobPtr)) {
        size = strlen(lpszNewValue);
        for(i=0; (i < size) && (i < (GlobPtr->datasize-1)); i++) {
            GlobPtr->data.Byte[i] = lpszNewValue[i];
        }
        GlobPtr->data.Byte[i] = '\0';

        // notify controls on list of change
        //if(GlobPtr->notify !=0)
            SendNotify(GlobPtr,lpView,IDVALUE,0);

        //SetModifiedFlag();
    }
}

void CGlobCtrl::Insert(long value, long index)
{
    // TODO: Add your dispatch handler code here
    LPBYTE source;
    LPBYTE dest;
    LONG size;
    long datacount;

    // exit if index is beyond range or Glob isnt setup
    if (!GlobPtr) return;
    datacount = GlobPtr->dim1 * GlobPtr->dim2 * GlobPtr->eltsize;
    if (((index * GlobPtr->eltsize) >= datacount) || (index < 0))
        return;

    if ((GlobPtr) && !(lpView->ReadOnly)) {
        source = GlobPtr->data.Byte + index * GlobPtr->eltsize;
        dest = source + GlobPtr->eltsize;
        size = datacount - (index+1) * GlobPtr->eltsize;

        //move data up (memmove handles overlapping memory regions)
        memmove(dest,source,size);

        //insert new data element
    }
}

```



```

switch (GlobPtr->eltsize)
{
    case 4: GlobPtr->data.Long[index] = value;
            break;
    case 2: GlobPtr->data.Short[index] = (short)value;
            break;
    default: GlobPtr->data.Byte[index] = (BYTE) value;
            break;
}
// notify controls on list of change
//if(GlobPtr->notify !=0)
    SendNotify(GlobPtr,lpView,IDVALUE,0);
}
}

```

```

BSTR CGlobCtrl::GetFullPath()

```

```

{
    CString strResult;
    // TODO: Add your property handler here
    strResult = m_FullPath;
    return strResult.AllocSysString();
}

```

```

void CGlobCtrl::SetFullPath(LPCTSTR lpszNewValue)

```

```

{
    // TODO: Add your property handler here
    SetModifiedFlag();
}

```

```

long CGlobCtrl::ResizeMMF(long NewSize)

```

```

{
    // TODO: Add your dispatch handler code here
    return MMFRemapView(NewSize);
}

```

```

long CGlobCtrl::SyncFileSize()

```

```

{
    if (m_FileSize != lpView->Size) {
        // mapviews are not synchronized so remap
        MMFRemapView(lpView->Size);
    }
    return 0;
}

```

```

void LogErrorString(CString errstr)

```

```

{
    if (!LOG_ERRORS) return;

    FILE *f;

    f = fopen("C:\\GlobErr.Log","a");

```

```

    fwrite(errstr,1,errstr.GetLength(),f);
    fclose(f);
}

```

```

void CGlobCtrl::SendNotifyX(short NotifyID = 0, short Value = 0)
{

```

```

    // TODO: Add your dispatch handler code here

```

```

    int i,mapIndex,bitIndex;
    int results;
    HWND mHwnd;
    tGlobMsg msg;
    long* tempmsg;
    long GlobID;

```

```

    // set up message to send for notify

```

```

    msg.PropID = NotifyID;

```

```

    msg.Value = Value;

```

```

    tempmsg = (long*)&msg;

```

```

    GlobID = (long)GlobPtr - (long)lpView;

```

```

    mHwnd = GetSafeHwnd();

```

```

    if(GlobPtr) {

```

```

        for (mapIndex=0;mapIndex<lpView->nNotifyMaps;mapIndex++){

```

```

            for (bitIndex = 0;bitIndex<32;bitIndex++){

```

```

                i = mapIndex*32+bitIndex;

```

```

                if((GetNotifyList(mapIndex) & (1<<bitIndex)) && (mHwnd != lpView->NotifyHandle[i])) {

```

```

                    results = ::PostMessage(lpView->NotifyHandle[i],USER_VALUECHANGED,*tempmsg,GlobID);

```

```

                    if (!results) // if the handle is invalid then remove it from the list

```

```

                        RemoveNotify(GlobPtr,lpView->NotifyHandle[i]);

```

```

                }

```

```

            }

```

```

        }

```

```

    }

```

```

}

```

```

BOOL CGlobCtrl::GetAutoSendNotify()
{

```

```

    // TODO: Add your property handler here

```

```

    return m_AutoNotify;
}

```

```

void CGlobCtrl::SetAutoSendNotify(BOOL bNewValue)
{

```

```

    // TODO: Add your property handler here

```

```

    m_AutoNotify = bNewValue;

```

```

    SetModifiedFlag();
}

```

```

BOOL CGlobCtrl::OnSetExtent(LPSIZEL lpSizeL)
{
    // TODO: Add your specialized code here and/or call the base class

    return false; //COleControl::OnSetExtent(lpSizeL);
}

long CGlobCtrl::GetNHandles()
{
    // TODO: Add your property handler here

    return m_MaxLinks;
}

void CGlobCtrl::SetNHandles(long nNewValue)
{
    // TODO: Add your property handler here
    SetModifiedFlag();
}

long CGlobCtrl::GetNNotifyMaps()
{
    // TODO: Add your property handler here
    if (lpView)
        return lpView->nNotifyMaps;

    return 0;
}

void CGlobCtrl::SetNNotifyMaps(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

BOOL CGlobCtrl::FormatMMF(long NotifyLimit)
{
    // TODO: Add your dispatch handler code here
    int x;

    if(lpView){
        if(lpView->RefCount > 1){
            ThrowError(CTL_E_PERMISSIONDENIED,"Sharing violation. Can not reformat MMF.");
            return false;
        }
    }
}

```

```

    if (NotifyLimit < 32) NotifyLimit = 32;
    x = (NotifyLimit-1)/32 + 1;
    lpView->nNotifyMaps = x;
    m_MaxLinks = x*32;

    m_FileSize += (m_MaxLinks*sizeof(long) + 4095);    // calc new size of file
    m_FileSize &= ~4095;                                // make size a multiple of 4096

    MMFErase();
    if(MMFCreate() == OK) return TRUE;
}
return FALSE;
}

long CGlobCtrl::GetGlobSize()
{
    // TODO: Add your property handler here

    if(GlobPtr) return GlobPtr->size;
    return sizeof(tGlob);
}

void CGlobCtrl::SetGlobSize(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

long CGlobCtrl::GetRefCount()
{
    // TODO: Add your property handler here
    if(lpView) return lpView->RefCount;

    return 0;
}

void CGlobCtrl::SetRefCount(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

BSTR CGlobCtrl::GetVersion()
{
    CString strResult;
    // TODO: Add your property handler here
    strResult = VERSION;
    return strResult.AllocSysString();
}

```

```

void CGlobCtrl::SetVersion(LPCTSTR lpszNewValue)
{
    // TODO: Add your property handler here

```

```

    SetModifiedFlag();
}

```

```

BSTR CGlobCtrl::GetUOM()
{

```

```

    CString strResult;

```

```

    // TODO: Add your property handler here

```

```

    tGlob* temp;

```

```

    if ((GlobPtr) && (GlobPtr->UOM) && lView){
        temp = (tGlob*)((int)lView + GlobPtr->UOM);
        strResult = temp->name;
    } else {

```

```

        strResult = "";
    }

```

```

    return strResult.AllocSysString();
}

```

```

void CGlobCtrl::SetUOM(LPCTSTR lpszNewValue)
{

```

```

    // TODO: Add your property handler here

```

```

    int uom;

```

```

    uom = MMFGetGlobIx(lpszNewValue);

```

```

    if (GlobPtr) GlobPtr->UOM = uom;

```

```

    SetModifiedFlag();
}

```

```

BSTR CGlobCtrl::GetLink()
{

```

```

    CString strResult;

```

```

    // TODO: Add your property handler here

```

```

    tGlob* temp;

```

```

    if ((GlobPtr) && (GlobPtr->link) && lView){
        temp = (tGlob*)((int)lView + GlobPtr->link);
        strResult = temp->name;
    } else {

```

```

        strResult = "";
    }

```

```

    return strResult.AllocSysString();
}

```

```

void CGlobCtrl::SetLink(LPCTSTR lpszNewValue)

```

```

{
    // TODO: Add your property handler here
    int link;
    CString newval;

    if (lpszNewValue != NULL)
        newval = lpszNewValue;
    else
        newval = "";

    link = MMFGetGlobIx(newval);
    if (GlobPtr) GlobPtr->link = link;
    SetModifiedFlag();
}

long CGlobCtrl::IndexOf(LPCTSTR GlobName)
{
    // TODO: Add your dispatch handler code here
    long index;

    index = MMFGetGlobIx(GlobName);
    if (index) index = -1;
    return index;
}

BOOL CGlobCtrl::GetNotifyOnChange(LPCTSTR GlobName)
{
    // TODO: Add your property handler here
    tGlob* GPtr;
    HWND hWind;
    long index;
    long mapIndex;
    long bitIndex;
    BOOL ret;
    BYTE* temp;
    long* MapPtr;

    hWind = GetSafeHwnd();
    ret = false;

    if (!lpView) return false;

    if (GlobName[0] == '\\0') {
        GPtr = GlobPtr;
    } else {
        GPtr = (tGlob*)MMFGetGlobPtr(GlobName);
    }

    if (GPtr && GPtr->ptrMap) {
        index = FindHandle(hWind);
        if (index != -1) {
            mapIndex = index/32;

```

```

        bitIndex = index - mapIndex*32;
        temp = (BYTE*)GPtr + GPtr->ptrMap;
        MapPtr = (long*)temp + mapIndex;
        if (*MapPtr & (1<<bitIndex)) ret = true;
    }
}

return ret;
}

void CGlobCtrl::SetNotifyOnChange(LPCTSTR GlobName, BOOL bNewValue)
{
    // TODO: Add your property handler here

    BOOL ret;
    HWND hWind;
    tGlob* GPtr;

    if (GlobName[0] == '\\0'){
        GPtr = GlobPtr;
    } else {
        GPtr = (tGlob*)MMFGetGlobPtr(GlobName);
    }

    if (GPtr) {
        CSingleLock LockMe(GlobLock);

        if (!AmbientUserMode()) {
            ThrowError(CTL_E_PERMISSIONDENIED, "This property can only be set at runtime.", 0);
            return;
        }

        LockMe.Lock(); // waits infinitely for resource to be available.
                       // can use a timeout value as a parameter (ms) if desired.

        hWind = GetSafeHwnd();

        if (bNewValue) {
            ret = AddNotify(GPtr, hWind);
        } else {
            ret = RemoveNotifyX(GPtr, hWind);
        }

        LockMe.Unlock();
    }

    SetModifiedFlag();
}

long CGlobCtrl::SetVisible()
{

```

```

HRESULT hresult;
IDispatch FAR* pdisp = (IDispatch FAR*)NULL;
DISPID dispid;
OLECHAR FAR* szVisible = L"Visible";
OLECHAR FAR* szTabStop = L"TabStop";
DISPPARAMS disparams;
DISPID MyDispid = DISPID_PROPERTYPUT;
VARIANTARG myarg[1];

```

```

disparams.rgvarg = myarg;
disparams.rgvarg[0].vt = VT_BOOL;
disparams.rgvarg[0].boolVal = FALSE; //MFC help says this fieldname is actually "bool"... yea right!
disparams.rgdispidNamedArgs = &MyDispid;
disparams.cArgs = 1;
disparams.cNamedArgs = 1;

```

```

pdisp = GetExtendedControl();
hresult = DISP_E_UNKNOWNINTERFACE;
if (pdisp) {
    //set visible to false
    hresult = pdisp->GetIDsOfNames(IID_NULL,&szVisible,1,LOCALE_USER_DEFAULT,&dispid);
    if (hresult == S_OK) {
        hresult = pdisp->Invoke(dispid,IID_NULL,LOCALE_USER_DEFAULT,DISPATCH_PROPERTYPUT,
            &disparams,NULL,NULL,NULL);
    }
    //set TabStop to false
    hresult = pdisp->GetIDsOfNames(IID_NULL,&szTabStop,1,LOCALE_USER_DEFAULT,&dispid);
    if (hresult == S_OK) {
        hresult = pdisp->Invoke(dispid,IID_NULL,LOCALE_USER_DEFAULT,DISPATCH_PROPERTYPUT,
            &disparams,NULL,NULL,NULL);
    }

    pdisp->Release();
}
return (long)hresult;
}

```



```
// Copyright 1998, 1999 SPX Corporation
#ifdef(AFX_GLOB_H__5F20D2DC_788C_11D1_9A9B_020701045A6B__INCLUDED_)
#define AFX_GLOB_H__5F20D2DC_788C_11D1_9A9B_020701045A6B__INCLUDED_
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
// Glob.h : main header file for GLOB.DLL
```

```
#if !defined( __AFXCTL_H__ )
#error include 'afxctl.h' before including this file
#endif
```

```
#include "resource.h" // main symbols
```

```
////////////////////////////////////
// CGlobApp : See Glob.cpp for implementation.
```

```
#define Glob_NAME_LENGTH 16
#define DEF_NOTIFYMAPS 8
#define NOTIFYMASK -1
#define OK 0
#define ERR_CANT_CREATE_FILE_MAPPING -2
#define ERR_CANT_MAP_VIEW_OF_FILE -1
#define ERR_INVALID_Glob_REFERENCE -3
#define ERR_INVALIDFILENAME 999
#define MEM_ALLOC 4096
#define IMAX_STRING 256
#define MMF_INTERCOM_MMF __TEXT("MMF_INTERCOM")
```

```
// FLAG VALUES TO USE IN MMFGETGlobPARAM AND MMFSETGlobPARAM
// USE ACTUAL BYTE OFFSETS FOR FASTER ACCESS
```

```
#define Glob_DIM2 4
#define Glob_DIM1 6
#define GlobELTSIZE 8
#define Glob_TYPE 10
#define Glob_PARAM 12 // addl data
#define Glob_DATASIZE 14 // addl
typedef struct
{
    long Size;
    int FirstGlob;
    int NextAvail;
    int ReadOnly; // is MMF Readonly right now?
    int RefCount;
    int nNotifyMaps;
    int Data[ 10 ]; // spare
    HWND NotifyHandle[0]; // hwnd for windows to notify of changes
} tControl;
```

```

typedef struct
{
    int size;
    BYTE name[ Glob_NAME_LENGTH ];
    short dim2;    // 2nd dimension
    short dim1;    // 1st dimension
    short eltsize; // byte size of each array element
    short type;    // type of array element    ..
    short extra;   // addl data. Waveforms use for Actual Length, etc.
    short command; // command to the device
    short status;  // status from the device
    short datasize; // addl
    long UOM;      // unit of measure link, if any
    long link;     // offset of parameter Glob, if any
    //long notifymap[DEF_NOTIFYSAPS]; // bitmap used to indicate who to notify if changed
    long ptrMap;
    union
    {
        long Long[0];
        short Short[0];
        BYTE Byte[0];
    } data;
} tGlob;

typedef struct
{
    short PropID;
    short Value;
} tGlobMsg;

class CGlobApp : public COleControlModule
{
public:
    BOOL InitInstance();
    int ExitInstance();
};

extern const GUID CDECL _tlid;
extern const WORD _wVerMajor;
extern const WORD _wVerMinor;

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_GLOB_H__5F20D2DC_788C_11D1_9A9B_020701045A6B__INCLUDED)

```

```
// Copyright 1998, 1999 SPX Corporation
// GlobPpg.cpp : Implementation of the CGlobPropPage property page class.
```

```
#include "stdafx.h"
#include "Glob.h"
#include "GlobPpg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
IMPLEMENT_DYNCREATE(CGlobPropPage, COlePropertyPage)
```

```
////////////////////////////////////
```

```
// Message map
```

```
BEGIN_MESSAGE_MAP(CGlobPropPage, COlePropertyPage)
```

```
//{{AFX_MSG_MAP(CGlobPropPage)
```

```
// NOTE - ClassWizard will add and remove message map entries
```

```
// DO NOT EDIT what you see in these blocks of generated code !
```

```
//}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
```

```
// Initialize class factory and guid
```

```
IMPLEMENT_OLECREATE_EX(CGlobPropPage, "GLOB.GlobPropPage.1",
```

```
0x5f20d2d7, 0x788c, 0x11d1, 0x9a, 0x9b, 0x2, 0x7, 0x1, 0x4, 0x5a, 0x6b)
```

```
////////////////////////////////////
```

```
// CGlobPropPage::CGlobPropPageFactory::UpdateRegistry -
```

```
// Adds or removes system registry entries for CGlobPropPage
```

```
BOOL CGlobPropPage::CGlobPropPageFactory::UpdateRegistry(BOOL bRegister)
```

```
{
    if (bRegister)
        return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
            m_clsid, IDS_GLOB_PPG);
    else
        return AfxOleUnregisterClass(m_clsid, NULL);
}
```

```
////////////////////////////////////
```

```
// CGlobPropPage::CGlobPropPage - Constructor
```

```

CGlobPropPage::CGlobPropPage() :
    COlePropertyPage(IDD, IDS_GLOB_PPG_CAPTION)
{
   //{{AFX_DATA_INIT(CGlobPropPage)
    // NOTE: ClassWizard will add member initialization here
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_DATA_INIT
}

////////////////////////////////////
// CGlobPropPage::DoDataExchange - Moves data between page and properties

void CGlobPropPage::DoDataExchange(CDataExchange* pDX)
{
   //{{AFX_DATA_MAP(CGlobPropPage)
    // NOTE: ClassWizard will add DDP, DDX, and DDV calls here
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}

////////////////////////////////////
// CGlobPropPage message handlers

```

```

// Copyright 1998, 1999 SPX Corporation
#ifdef AFX_GLOBCTL_H__5F20D2E4_788C_11D1_9A9B_020701045A6B__INCLUDED_
#define AFX_GLOBCTL_H__5F20D2E4_788C_11D1_9A9B_020701045A6B__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// GlobCtl.h : Declaration of the CGlobCtrl ActiveX Control class.

////////////////////

// CGlobCtrl : See GlobCtl.cpp for implementation.
#include <afxmt.h>
#include <memory.h>
#include <string.h>

#define IDVALUE 1
#define IDSTATUS 2
#define IDCOMMAND 3
#define VERSION "1.2i"

class CGlobCtrl : public COleControl
{
    DECLARE_DYNCREATE(CGlobCtrl)

// Constructor
public:
    CGlobCtrl();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CGlobCtrl)
public:
    virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);
    virtual void DoPropExchange(CPropExchange* pPX);
    virtual void OnResetState();
    virtual void OnFinalRelease();
    virtual BOOL OnSetExtent(LPSIZEL lpSizeL);
//}}AFX_VIRTUAL

// Implementation
protected:
    ~CGlobCtrl();

    BEGIN_OLEFACTORY(CGlobCtrl)
        virtual BOOL VerifyUserLicense();
        virtual BOOL GetLicenseKey(DWORD, BSTR FAR*);
    END_OLEFACTORY(CGlobCtrl)

    DECLARE_OLETYPELIB(CGlobCtrl) // GetTypeInfo
    DECLARE_PROPPAGEIDS(CGlobCtrl) // Property page IDs
    DECLARE_OLECTLTYPE(CGlobCtrl) // Type name and misc status

```

```

// Message maps
//{{AFX_MSG(CGlobCtrl)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

// Dispatch maps
//{{AFX_DISPATCH(CGlobCtrl)
afx_msg long GetGlobIndex();
afx_msg void SetGlobIndex(long nNewValue);
afx_msg long GetDim1Size();
afx_msg void SetDim1Size(long nNewValue);
afx_msg long GetDim2Size();
afx_msg void SetDim2Size(long nNewValue);
afx_msg long GetElementSize();
afx_msg void SetElementSize(long nNewValue);
afx_msg long GetType();
afx_msg void SetType(long nNewValue);
afx_msg long GetExtra();
afx_msg void SetExtra(long nNewValue);
afx_msg long GetDataSize();
afx_msg void SetDataSize(long nNewValue);
afx_msg BSTR GetGlobName();
afx_msg void SetGlobName(LPCTSTR lpszNewValue);
afx_msg long GetFileSize();
afx_msg void SetFileSize(long nNewValue);
afx_msg BSTR GetFileName();
afx_msg void SetFileName(LPCTSTR lpszNewValue);
afx_msg long GetStatus();
afx_msg void SetStatus(long nNewValue);
afx_msg long GetCommand();
afx_msg void SetCommand(long nNewValue);
afx_msg long GetGlobPtr();
afx_msg void SetGlobPtr(long nNewValue);
afx_msg long GetDataPtr();
afx_msg void SetDataPtr(long nNewValue);
afx_msg long GetAvailSize();
afx_msg void SetAvailSize(long nNewValue);
afx_msg BOOL GetReadOnlyMMF();
afx_msg void SetReadOnlyMMF(BOOL bNewValue);
afx_msg BOOL GetNotify();
afx_msg void SetNotify(BOOL bNewValue);
afx_msg short GetByteValue();
afx_msg void SetByteValue(short nNewValue);
afx_msg long GetLValue();
afx_msg void SetLValue(long nNewValue);
afx_msg short GetIValue();
afx_msg void SetIValue(short nNewValue);
afx_msg BSTR GetStrValue();
afx_msg void SetStrValue(LPCTSTR lpszNewValue);
afx_msg BSTR GetFullPath();
afx_msg void SetFullPath(LPCTSTR lpszNewValue);

```

```

afx_msg BOOL GetAutoSendNotify();
afx_msg void SetAutoSendNotify(BOOL bNewValue);
afx_msg long GetNHandles();
afx_msg void SetNHandles(long nNewValue);
afx_msg long GetNNotifyMaps();
afx_msg void SetNNotifyMaps(long nNewValue);
afx_msg long GetGlobSize();
afx_msg void SetGlobSize(long nNewValue);
afx_msg long GetRefCount();
afx_msg void SetRefCount(long nNewValue);
afx_msg BSTR GetVersion();
afx_msg void SetVersion(LPCTSTR lpszNewValue);
afx_msg BSTR GetUOM();
afx_msg void SetUOM(LPCTSTR lpszNewValue);
afx_msg BSTR GetLink();
afx_msg void SetLink(LPCTSTR lpszNewValue);
afx_msg long MMFClose();
afx_msg long MMFAddGlob(LPCTSTR GlobName, long UnitsIndex, long Dim2Size, long Dim1Size, long ElementSize, long
Type, long Extra);
afx_msg long MMFAddGlobEx(LPCTSTR GlobName, LPCTSTR UnitsName, long Dim2Size, long Dim1Size, long ElementSize,
ong Type, long Extra);
afx_msg BOOL GetFirstGlob();
afx_msg BOOL GetNextGlob();
afx_msg void Erase();
afx_msg void MMFErase();
afx_msg void Insert(long value, long index);
afx_msg long ResizeMMF(long NewSize);
afx_msg void SendNotifyX(short NotifyID, short Value);
afx_msg BOOL FormatMMF(long NotifyLimit);
afx_msg long IndexOf(LPCTSTR GlobName);
afx_msg long GetValue(long Dim2, long Dim1);
afx_msg void SetValue(long Dim2, long Dim1, long nNewValue);
afx_msg long GetNotifyHandle(short index);
afx_msg void SetNotifyHandle(short index, long nNewValue);
afx_msg short GetAbValue(long index);
afx_msg void SetAbValue(long index, short nNewValue);
afx_msg long GetAiValue(long index);
afx_msg void SetAiValue(long index, long nNewValue);
afx_msg short GetAiValue(long index);
afx_msg void SetAiValue(long index, short nNewValue);
afx_msg long GetValueSD(long n);
afx_msg void SetValueSD(long n, long nNewValue);
afx_msg long GetNotifyList(long index);
afx_msg void SetNotifyList(long index, long nNewValue);
afx_msg BOOL GetNotifyOnChange(LPCTSTR GlobName);
afx_msg void SetNotifyOnChange(LPCTSTR GlobName, BOOL bNewValue);
/}}AFX_DISPATCH
DECLARE_DISPATCH_MAP()

afx_msg void AboutBox();
afx_msg long OnValueChanged(UINT, LONG);

```

```

// Event maps
//{{AFX_EVENT(CGlobCtrl)
void FireChange(short PropID, short Value, long SendID)
    {FireEvent(eventidChange,EVENT_PARAM(VTS_I2 VTS_I2 VTS_I4), PropID, Value, SendID);}
//}}AFX_EVENT
DECLARE_EVENT_MAP()

```

```

// Dispatch and event IDs

```

```

public:

```

```

enum {
//{{AFX_DISP_ID(CGlobCtrl)
dispidGlobIndex = 1L,
dispidDim1Size = 2L,
dispidDim2Size = 3L,
dispidElementSize = 4L,
dispidType = 5L,
dispidExtra = 6L,
dispidDataSize = 7L,
dispidGlobName = 8L,
dispidFileSize = 9L,
dispidFileName = 10L,
dispidStatus = 11L,
dispidCommand = 12L,
dispidGlobPtr = 13L,
dispidDataPtr = 14L,
dispidAvailSize = 15L,
dispidReadOnlyMMF = 16L,
dispidNotify = 17L,
dispidValue8 = 18L,
dispidValue32 = 19L,
dispidValue16 = 20L,
dispidStrValue = 21L,
dispidFullPath = 22L,
dispidAutoSendNotify = 23L,
dispidNHandles = 24L,
dispidNNotifyMaps = 25L,
dispidGlobSize = 26L,
dispidRefCount = 27L,
dispidVersion = 28L,
dispidUOM = 29L,
dispidLink = 30L,
dispidValue = 43L,
dispidCloseMMF = 31L,
dispidAddNew = 32L,
dispidAddNewEx = 33L,
dispidGetFirstGlob = 34L,
dispidGetNextGlob = 35L,
dispidErase = 36L,
dispidEraseMMF = 37L,
dispidNotifyHandle = 44L,
dispidAValue8 = 45L,
dispidAValue32 = 46L,

```



```

dispidAValue16 = 47L,
dispidValueSD = 48L,
dispidInsert = 38L,
dispidResizeMMF = 39L,
dispidSendNotify = 40L,
dispidNotifyMap = 49L,
dispidFormatMMF = 41L,
dispidIndexOf = 42L,
dispidNotifyOnChange = 50L,
eventidChange = 1L,
//}}AFX_DISP_ID
};

private:
    tGlob * GlobPtr;
    long datasize;
    CString m_GlobName;
    BOOL m_Notify;
    CMutex *GlobLock;
    CMutex *MMFLock;
    CString m_FullPath;
    CString m_FileName;
    int m_FileSize;
    HANDLE f;
    HANDLE hFileMapT;
    HANDLE s_hFileMap;
    tControl *lpView;
    LPBYTE lpLast;
    BOOL m_AutoNotify;
    long m_MaxLinks;

    CString SPX_NOTIFY_MUTEX;
    CString SPX_MMF_MUTEX;

    //private member functions
    long MMFCreate(void);
    long MMFOpen(LPCTSTR, long, LPCTSTR, long);
    long MMFRemapView( long );
    long MMFGetGlobPtr(LPCTSTR);
    tGlob *MMFNextAvailGlob( long );
    tGlob *MMFFirstGlob();
    void MMFEraseGlob( tGlob *);
    long MMFGetGlobIx(LPCTSTR);
    void MMFResetGlobs(void);
    BOOL AddNotify(tGlob*,HWND);
    BOOL RemoveNotify(tGlob*,HWND);
    int FindHandle(HWND);
    void SendNotify(tGlob*, tControl*, short, short);
    void MMFClearGlobBits(long BitMap);
    long SyncFileSize();
    void SetBitMap(tGlob*,long index);
    BOOL RemoveNotifyX(tGlob* GlobPtr, HWND my_hWnd);
    long MMFClearGlobBit(tGlob* GPtr,long index);

```



```

// Copyright 1998, 1999 SPX Corporation
#ifdef AFX_STDAFX_H__5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_
#define AFX_STDAFX_H__5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// stdafx.h : include file for standard system include files,
//           or project specific include files that are used frequently,
//           but are changed infrequently

#define VC_EXTRALEAN    // Exclude rarely-used stuff from Windows headers

#include <afxctl.h>      // MFC support for ActiveX Controls

// Delete the two includes below if you do not wish to use the MFC
// database classes
#include <afxdb.h>       // MFC database classes
#include <afxdao.h>      // MFC DAO database classes

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__5F20D2DA_788C_11D1_9A9B_020701045A6B__INCLUDED_)

```

```
// Copyright 1998, 1999 SPX Corporation
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Glob.rc
//
```

```
#define IDS_GLOB 1
#define IDD_ABOUTBOX_GLOB 1
#define IDB_GLOB 1
#define IDI_ABOUTDLL 1
#define IDS_GLOB_PPG 2
#define IDS_GLOB_PPG_CAPTION 200
#define IDD_PROPPAGE_GLOB 200
```

```
// Next default values for new objects
```

```
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 203
#define _APS_NEXT_COMMAND_VALUE 32768
#define _APS_NEXT_CONTROL_VALUE 201
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
```

```

// Copyright 1998, 1999 SPX Corporation
#ifdef AFX_GLOBPPG_H__5F20D2E6_788C_11D1_9A9B_020701045A6B__INCLUDED_
#define AFX_GLOBPPG_H__5F20D2E6_788C_11D1_9A9B_020701045A6B__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// GlobPpg.h : Declaration of the CGlobPropPage property page class.

////////////////////

// CGlobPropPage : See GlobPpg.cpp.cpp for implementation.

class CGlobPropPage : public COlePropertyPage
{
    DECLARE_DYNCREATE(CGlobPropPage)
    DECLARE_OLECREATE_EX(CGlobPropPage)

// Constructor
public:
    CGlobPropPage();

// Dialog Data
   //{{AFX_DATA(CGlobPropPage)
    enum { IDD = IDD_PROPPAGE_GLOB };
    // NOTE - ClassWizard will add data members here.
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

// Message maps
protected:
   //{{AFX_MSG(CGlobPropPage)
    // NOTE - ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_GLOBPPG_H__5F20D2E6_788C_11D1_9A9B_020701045A6B__INCLUDED)

```

PrintUtilities - 1

```
Sub StartPrinting(TemplateName As String)
    Dim OldRegSection As String

    'Setup printer
    With frmATPPrint.registry1
        OldRegSection = .Section
        .Section = "Printer"
        .Value("Command") = "Print"
        .Value("Template") = TemplateName
        .Section = OldRegSection
    End With
    'Start printing process
    frmATPPrint.PrintControl.Command = GL_ModPrinterRequest
End Sub
```

PrintUtilities - 1